ESP-Techpedia_EN



Release master Espressif Systems Jul 03, 2025

Table of contents

Table of contents	Tab	ole	of	contents
-------------------	-----	-----	----	----------

1	ESP-	FAQ	3	;
2	ESP	Friends	5	;
	2.1	Get Star	rted	j
		2.1.1	Board selection	j
		2.1.2	Flash Programming 23	;
		2.1.3	Environment Setup)
		2.1.4	Getting Started with Project Development 55	j
	2.2	Advance	ed Development)
		2.2.1	Component Management and Usage 59)
		2.2.2	Advanced Code Debugging	2
		2.2.3	Performance Optimization	2
		2.2.4	Network Protocol Related	
		2.2.5	System Related	5
		2.2.6	Low-Power Bluetooth Application Note	j
	2.3	Introduc	ction to Application Solutions	;
		2.3.1	AI Solution	;
		2.3.2	Brushless Motor Solution	;
		2.3.3	Camera Solution	ŀ
		2.3.4	LCD Solution)
		2.3.5	ESP-CSI Solution)
		2.3.6	ESP-IoT-Bridge Solution	;
		2.3.7	ESP-NOW Solution	j
		2.3.8	Mesh Solutions	5
		2.3.9	Provisioning Solution	
		2.3.10	Thread Solution	ļ
		2.3.11	Zigbee Solution	ŀ
		2.3.12	Wireless Network Card Solution	5
		2.3.13	Lighting Solution	;
	2.4	Recom	nended Tools	ŀ
		2.4.1	Wireshark Packet Capture Tutorial on Windows	j

i

ESP-Techpedia is a comprehensive platform by Espressif Systems, which aggregates various technical documents with the aim to provide developers with comprehensive and accurate information about Espressif's technologies. We not only cover documents necessary for getting started with software development, but also collect common technical Q&A to address developers' queries when using Espressif's chips, modules, or development boards.

On ESP-Techpedia, you can easily access a wide range of Espressif technical documents. For instance, the ESP Friends documentation helps you quickly get started with Espressif development, while the ESP-FAQ provides solutions to common issues.

In addition to compiling a variety of technical documents, ESP-Techpedia warmly welcomes new document requests from developers worldwide. We hope to offer more valuable technical documents through close interaction, making development an effortless experience.

Whether you are a beginner or an experienced developer, ESP-Techpedia will be your indispensable technical reference, empowering you to achieve your project goals more efficiently.



Chapter 1



Please refer to ESP-FAQ

Chapter 2

ESP Friends

ESP Friends documents can help developers quickly get started with Espressif's development, such as guidance on how to select ESP hardware, build an environment, get started with software, etc. ESP solution recommendations and other advanced development documents will be added in the future, so stay tuned!

2.1 Get Started

In this section, we provide a basic getting started guide for ESP beginners. Whether you' re new to it or want to solidify the basics, this section will guide you through the basics of ESP operation. You will learn key basic knowledge, including hardware selection, firmware programming, environment setup, etc.

2.1.1 Board selection

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

In the rapidly changing IoT market, Espressif has launched a series of distinctive ESP chips to fully meet the evolving demands. Choosing the right ESP chip is particularly important as it will directly affect performance and functionality of the product. Choose the suitable ESP chip based on the project's **application scenario**, **power consumption**, **wireless communication**, **GPIO and memory requirements**.

To help developers better understand the ESP chips and modules, below is a brief comparison chart.

Chip	Year of	Application Sce-	Wireless	GPIO	SRAM	PSRAM Support
ESP32-P4	2025	A high-performance	N/A	55	768 КВ НР	
		connectivity, HMI,			L2MEM,	
		and excellent safety			32 KB LP	
		features, featuring			SRAM,	
		AI instruction ex-			8 KB	
		tensions, advanced			TCM	
		and integrated high-				
		speed peripherals.				
ESP32-C5	2025	The first RISC-V	2.4GHz &	29	384	✓
		SoC that supports	5GHz Wi-Fi		KB HP	
		2.4GHz & 5GHz	6 + BLE5.0		SRAM,	
		Bluetooth 5 (LE)	+ Illieau + Zighee		SRAM	
		and IEEE 802.15.4	Ligott		Sichin	
		(Zigbee, Thread),				
		specifically designed				
		for lot applica-				
		efficient wireless				
		transmission.				
ESP32-C6	2023	Ultra-low power IoT	BLE 5.0 + Wi-	23	512 KB	×
		devices with long	Fi 6 + Thread +			
		battery life, Thread	Zigbee			
		ter gateways, Zigbee				
		bridges				
ESP32-C2	2022	Sockets, lighting,	BLE 5.0 + Wi-	14	272 KB	x
		sensors, simple smart	Fi 4			
ESP32-H2	2021	Thread border	BLE 5.0 +	19	320 KB	\checkmark
		routers, Matter	Thread +			
		gateways, Zigbee	Zigbee			
	2020	bridges		26	512 VD	
ESP32-53	2020	smart cameras, face	BLE $5.0 + W_{1}$ - Fi 4	30	512 KB	✓
		recognition, voice	111			
		wake-up, real-time				
		data collection and				
		processing, complex				
ESP32-S2	2020	Real-time data col-	Wi-Fi 4	36	320 KB	\checkmark
		lection and process-				
		ing, complex periph-				
	2020	eral control	$\mathbf{D} \mathbf{E} 5 0 + \mathbf{W}$	15	400 KD	
ESP32-C3	2020	switch sockets smart	Fi 4	15	400 KD	X
		home appliances,				
		industrial control				
ECD22	2016	field		26	500 KD	
ESP32	2016	Recommended to	$\begin{array}{c} \mathbf{BLE} 4.2 + \mathbf{BI} + \\ \mathbf{Wi}_{1} \mathbf{Fi} 4 \end{array}$	20	520 KB	✓
		ESP32-S3	VV 1-1 1 T			
ESP8266	2014	Recommended to	Wi-Fi 4	11	160 KB	×
Forressif System-		use modern, more	6		ח	alansa maatar
Espressii Systems		secure, and powerful	o ment Feedback		K	clease master
		C2 or ESP32-C3				

Table 1: Chip Comparison

Note: The above is just a brief introduction to the ESP chip series. If you want to further understand the specific details and features of each series of chips or modules, you can use the ESP Chip & Module Selection Tool to easily obtain relevant informations. This tool will select the ESP chip most suitable for the developer's application based on project requirements and technical specifications.

Chips, Modules, Development Boards

Espressif officially provides chips, modules, and development boards, which have different uses and features in the development and deployment process of IoT applications.

1. Chip:



- The ESP series chip is the fundamental integrated circuit (IC) manufactured by Espressif, serving as the core component of the entire product line. These chips typically integrate processors (CPUs), memory, communication interfaces, General Purpose Input Output (GPIO), and other hardware functionalities. They can be directly integrated into custom circuit boards to create highly specialized IoT devices, making them suitable for projects demanding compact size and specific functionalities.
- The chips require connection to external components for power and functionality. Additionally, developing products with these chips involves certification for wireless communication protocols, which can be somewhat intricate.
- 2. Module:



- The module is a package of Espressif's chip, integrating the chip, crystal, antenna, and flash. Espressif's modules usually have FCC, CE, and other certifications, allowing developers to focus more on the development of applications, without having to worry about the hardware matching design of the antenna and the details of RF certification, thus accelerating the speed of product launch.
- Compared to individual chips, modules offer greater convenience in hardware design and project development.
- 3. Development Board:



- The development board is a comprehensive development platform that integrates Espressif modules. It includes various interfaces and resources for debugging, development, and testing, and facilitates software debugging and firmware flashing during the development phase. Typically, in the early stages of project development, rapid testing and verification are conducted using the development board. As the product moves towards mass production, the module is integrated into the final design.
- Moreover, the development board serves as an essential tool for developers new to Espressif chips, enabling them to quickly familiarize themselves with the platform. It accelerates the verification of devel-

opers' ideas and designs, enabling rapid prototyping.

Selection Guide Choosing the right chip, module, or development board depends on the project's requirements, timeline, technical capabilities, and budget. Here are some factors to consider when choosing:

- 1. Rapid development and prototype verification:
- **Development boards** are highly beneficial for rapid function development and verification in the early stages of the project
- 2. Custom hardware design:
 - If a highly customized circuit board and hardware design are required, **chips** are a more suitable choice

Note: Custom designs require certification through wireless communication protocols, which may increase development time and cost.

- 3. Speed to market:
- **Modules** usually can speed up the product's time-to-market, with comprehensive radio frequency certification reports. Developers can focus more on the development of applications, without having to deal with the details of antenna hardware matching design and radio frequency certification.
- 4. Cost budget:
- Using **chips** typically costs less, but custom design may increase time and development difficulty. Modules have a relatively higher cost, but can speed up the development process
- 5. Team technical capabilities:
- If your team is new to Espressif chips or has limited technical resources, using **modules** is easier to get started, accelerates the project process and **reduces technical risks**. Using chips requires higher technical capabilities and more development experience

ESP32-P4

Supported functions:

- 55 programmable GPIO pins (QFN10*10), supporting MIPI-CSI (Integrated Image Signal Processor ISP) and MIPI-DSI interfaces, USB OTG 2.0 HS, Ethernet, and SDIO Host 3.0.
- Available for development solutions:

Development board

• ESP32-P4-Function-EV-Board : ESP32-P4-Function-EV-Board is a multimedia development board based on the ESP32-P4 chip.



• ESP32-P4-EYE : ESP32-P4-EYE is a vision development board designed for camera applications.



Hardware Design Guide

• ESP32-P4 Hardware Design Guidelines

Purchase link:

• ESP32-P4 Development Board

ESP32-C5

Supported functions:

- 29 programmable GPIO pins (QFN6*6), supporting SDIO, SPI, UART, I2C, I2S, RMT, TWAI, and PWM.
- Suitable for development schemes: Ultra-low power IoT devices with long-lasting battery life, 2.4GHz & 5GHz WiFi 6, Thread border routers, Matter gateways, Zigbee bridges.

Development Board

• ESP32-C5-DevKitC-1 : ESP32-C6-DevKitC-1 is a beginner-level development board that can be used to burn and experience examples in IDF.



Hardware Design Guide

• ESP32-C5 Hardware Design Guide

Purchase link:

• ESP32-C5 Development Board

ESP32-C6

Supported Features:

- 30 (QFN40) or 22 (QFN32) programmable GPIO pins, supporting SPI, UART, I2C, I2S, RMT, TWAI, and PWM
- Can be used for development solutions: ultra-low power IoT devices with long battery life, Thread border routers, Matter gateways, Zigbee bridges

Development Boards

• ESP32-C6-DevKitC-1 : ESP32-C6-DevKitC-1 is an entry-level development board that can be used to flash and experience examples in IDF.



• ESP32-C6-DevKitM-1 : ESP32-C6-DevKitCM-1 is an entry-level development board that can be used to flash and experience examples in IDF.



Hardware Design Guide

• ESP32-C6 Hardware Design Guide

Purchase Link:

• ESP32-C6 Development Board

ESP32-C2

Supported Features:

- 14 programmable GPIO pins: SPI, UART, I2C, LED PWM controller, SAR ADC/DAC, temperature sensor
- · Can be used for development schemes: sockets, lighting, sensors, simple smart home appliances

Development Board

• ESP8684-DevKitM-1 : ESP8684-DevKitM-1 is an entry-level development board that can be used to flash and experience examples in IDF.



Hardware Design Guide

• ESP32-C2 Hardware Design Guide

Purchase Link:

• ESP32-C2 Development Board

ESP32-H2

Supported Features:

- 19 programmable GPIO pins, supporting common peripheral interfaces such as UART, SPI, I2C, I2S, infrared transceiver, LED PWM, full-speed USB serial/JTAG controller, GDMA, MCPWM
- Can be used for development schemes: Thread border router, Matter gateway, Zigbee bridge

Development Board

• ESP32-H2-DevKitM-1 : ESP32-H2-DevKitM-1 is an entry-level development board that can be used to provision and experience examples in IDF.



Hardware Design Guide

• ESP32-H2 Hardware Design Guide

Purchase Link:

• ESP32-H2 Development Board

ESP32-S3

Supported Features:

- Common peripheral interfaces such as SPI, I2S, I2C, PWM, RMT, ADC, UART, SD/MMC host controller, and TWAI controller, etc.
- Can be used for development schemes: Smart camera, face recognition, voice recognition, voice wake-up, real-time data collection and processing, complex peripheral control

Development Board:

• EESP32-S3-DevKitC-1 : ESP32-S3-DevKitC-1 is an entry-level development board that can be used to provision and experience examples in IDF.

• ESP32-S3-DevKitM-1 : ESP32-S3-DevKitM-1 is a beginner-friendly development board that can be used to flash and experience examples in IDF.



• ESP32-S3-BOX : ESP-BOX provides users with a platform for developing and controlling smart home devices based on voice assistant + touch screen control, sensors, infrared controllers, and smart Wi-Fi gateways.



• ESP32-S3-EYE : ESP32-S3-EYE is a small AI (Artificial Intelligence) development board launched by Espressif. The development board is equipped with a 2-megapixel camera, an LCD display, and a microphone, suitable for applications such as image recognition and audio processing. You can use ESP-WHO to develop various AIoT (Artificial Intelligence of Things) applications, such as smart doorbells, monitoring systems, face recognition attendance machines, etc.



- ESP32-S3-USB-OTG : ESP32-S3-USB-OTG is a development board focused on USB-OTG function verification and application development, based on ESP32-S3 SoC, supports Wi-Fi and BLE 5.0 wireless functions, supports USB host and USB slave functions. It can be used to develop wireless storage devices, Wi-Fi network cards, LTE MiFi, multimedia devices, virtual keyboards and mice, and other applications.
- ESP32-S3-Korvo-1 : ESP32-S3-Korvo-1 is an AI development board launched by Espressif, equipped with



the ESP32-S3 chip and Espressif's voice recognition SDK ESP-Skainet. ESP32-S3-Korvo-1 supports voice wake-up and offline voice command recognition in both Chinese and English. You can use ESP-Skainet to develop various voice recognition applications, such as smart screens, smart plugs, smart switches, etc.



• ESP32-S3-Korvo-2 : ESP32-S3-Korvo-2 is a multimedia development board based on the ESP32-S3 chip, equipped with a dual microphone array, supporting voice recognition and near/far field voice wake-up. It also carries peripherals such as LCD, camera, microSD card, etc., and can support JPEG-based video stream processing, meeting the development needs of users for low-cost, low-power, and networked audio and video products.



Hardware Design Guide

• ESP32-S3 Hardware Design Guide

Purchase link:

• ESP32-S3 Development Board

ESP32-S2

Supported Features:

- Full-speed USB OTG interface, SPI, I2S, UART, I2C, LED PWM, LCD interface, Camera interface, ADC, DAC, touch sensor
- Can be used for development solutions: real-time data collection and processing, complex peripheral control

Development Boards:

• ESP32-S2-DevKitC-1 : ESP32-S2-DevKitC-1 is a beginner-level development board that can be used to flash and experience examples in IDF.



• ESP32-S2-HMI-DevKit-1 : ESP32-S2-HMI-DevKit-1 is designed for GUI application scenarios, capable of realizing smart home interactive panels, speakers with screens, alarm clocks, and other human-machine interactive interfaces for intelligent control. This development board has a wealth of onboard sensors and expansion interfaces, making it easy for users to quickly carry out secondary development and implement various functions.



• ESP32-S2-Saola-1 : ESP32-S2-Saola-1 is a small development board based on ESP32-S2 from Espressif, which can be used to flash and experience examples in IDF.



Hardware Design Guide

• ESP32-S2 Hardware Design Guide

Purchase link:

• ESP32-S2 Development Board

ESP32-C3

Supported Features:

- Rich communication interfaces and GPIO pins, supporting multiple external SPI, Dual SPI, Quad SPI, QPI flash
- Can be used for development solutions: electrical lighting, switch sockets, smart home appliances, industrial control fields

Development Board

• ESP32-C3-DevKitM-1 : ESP32-C3-DevKitM-1 is an entry-level development board, using the ESP32-C3-MINI-1 module, which is named for its small size. It can be used to flash and experience examples in IDF.



• ESP32-C3-DevKitC-02 : ESP32-C3-DevKitC-02 is an entry-level development board, which can be used to flash and experience examples in IDF.



- ESP32-C3-DevKit-RUST-1 : ESP32-C3-DevKit-RUST-1 is an entry-level development board, which can be used to flash and experience examples in IDF.
- ESP32-C3-AWS-ExpressLink-DevKit : ESP32-C3-AWS-ExpressLink-DevKit uses an abstract application programming interface (API) to connect any host application to AWS IoT Core and its services. It has the shape of an Arduino expansion board, so it can be directly plugged into a standard Arduino. It can also be used with Raspberry Pi or any other host.



Hardware Design Guide

• ESP32-C3 Hardware Design Guide

Purchase link:

• ESP32-C3 Development Board

ESP32

Supported Features:

- Provides multiple GPIO pins, including digital input/output, analog input, PWM output, I2C, SPI, UART, etc.
- Development plan: Recommend using the latest released ESP32-S3

Development Boards

- ESP32-DevKitC : ESP32-DevKitC V4 is a small development board based on ESP32, which can be used to flash and experience examples in IDF.
- ESP-EYE : ESP-EYE is a development board aimed at the face recognition and voice recognition market, equipped with a 200 W pixel camera, digital microphone, which can meet various AI application development needs. In addition, this development board also supports Wi-Fi image transmission, Micro USB debugging and power supply, can achieve voice wake-up, face detection and recognition functions, and can assist users in developing highly integrated AI solutions.
- ESP32-LyraT : ESP32-LyraT is designed for the audio application market. It provides an audio codec chip, onboard dual microphones, headphone output, two 3-watt speaker outputs, dual auxiliary inputs, and lithium battery charging management hardware support.
- In addition, there are seven other development boards in the ESP32 series for audio processing, but we recommend developers to use the latest ESP32-S3 series audio development boards.
- ESP32-LCDKit : ESP32-LCDKit is an HMI (Human-Machine Interaction) development board with ESP32-DevKitC as the core, which can be connected to an external screen and integrates peripherals such as SD-Card, DAC-Audio, mainly used for HMI related development and evaluation.

• ESP32-Ethernet-Kit : The ESP32-Ethernet-Kit is an Ethernet to Wi-Fi development board that can provide Wi-Fi connectivity for Ethernet devices. To provide more flexible power options, the ESP32-Ethernet-Kit also supports Power over Ethernet (PoE).

Hardware Design Guide

• ESP32 Hardware Design Guide

Purchase Link:

• ESP32 Development Board

ESP8266

Supported Features:

- Provides multiple GPIO pins that can be used for various purposes, such as UART, I2C, SPI, etc.
- Development Solution: It is recommended to use the latest released ESP32-C2 or ESP32-C3

Development Board

• ESP8266-DevKitC : The ESP8266-DevKitC is a compact ESP8266 development board that can be used to flash and experience examples in IDF.

Hardware Design Guide

• ESP8266 Hardware Design Guide

Purchase Link:

• ESP8266 Development Board

2.1.2 Flash Programming

This document shows the hardware and software environment that each ESP chip needs to meet when programming firmware.

Hardware Requirements for Entering Download Mode on Different ESP Chips

When using a chip or module, please select the corresponding ESP series chip for hardware wiring configuration.

ESP8266

The ESP8266 downloads firmware via UARTO (TX0 (GPIO1) and RXD (GPIO3)) by default.

Hardware Connection The following wiring conditions need to be satisfied:

```
VDD > 3V3
GND -> GND
EN -> Pull Up (Used for power on, can't be float)
GPIO0 -> Pull Down
GPIO15 -> Pull Down
TXD0(GPIO1) -> RX
RXD0(GPIO3) -> TX
```

Hardware conditions must be met

- ESP8266 chip working voltage range is 2.5 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.
- ESP8266 modules working voltage range is $2.7 \ V \sim 3.6 \ V$; if you use a single power supply, the recommended voltage of the power supply is $3.3 \ V$, and its recommended output current is $500 \ mA$ or more.

Reference Hardware wiring principle summary:

• Chinese Blog : Hardware environment requirements for ESP8266 firmware flashing

Official Documentation:

- ESP8266 Datasheet
- ESP8266 Hardware Design Guidelines

ESP32

The ESP32 downloads firmware via UARTO (TX0 (GPIO1) and RXD (GPIO3)) by default.

Hardware Connection The following wiring conditions need to be satisfied:

```
VDD
       -> 3V3
GND
       -> GND
ΕN
       -> Pull Up
                     (Used for power on Control, can't be float)
GPIOO
       -> Pull Down
                     (Enter download mode)
                     (Default is lw level)
GPIO2
       -> Pull Down
TXDO(GPIO1)
              -> RX
RXD0(GPIO3)
              -> TX
```

Note:

- The Strapping pin GIO12 (MTDI) is used for selecting the VDD_SPI Flash voltage.
 - When use the 1.8V VDD_SPI Flash, the GIO12 (MTDI) need pull up when chip power on.
 - When use the 3.3V VDD_SPI Flash, the GIO12 (MTDI) need pull down when chip power on.

Hardware conditions must be met

- ESP32 chip working voltage range is 2.3 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.
- ESP32 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.

Reference Hardware wiring principle summary:

• Chinese Blog : Hardware environment requirements for ESP32 firmware flashing

Official Documentation:

- ESP32 Datasheet
- ESP32 Hardware Design Guidelines

ESP32-S2

The ESP32-S2 supports two ways to download the firmware: UART0 and USB.

Hardware Connection When you are use the UART0 pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3

GND -> GND

EN -> Pull Up (Used for power on, can't be float)

GPIO0 -> Pull Down (Enter download mode)

GPIO46 -> Pull Down

TXD0(GPIO43) -> RX

RXD0(GPIO44) -> TX
```

When you are use the USB pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on, can't be float)
GPIO0 -> Pull Down (Enter download mode)
GPIO46 -> Pull Down (Default is Low level)
GPIO19 -> USB_D-
GPIO20 -> USB_D+
```

Note:

- The Strapping pin GIO45 is used for selecting the VDD_SPI Flash voltage.
 - When use the 1.8V VDD_SPI Flash, the GIO45 need pull up when chip power on.
 - When use the 3.3V VDD_SPI Flash, the GIO45 need pull down when chip power on.

Hardware conditions must be met

- ESP32-S2 chip working voltage range is 2.8 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.
- ESP32-S2 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.

Reference Hardware Wiring Principle Summary:

- Chinese blog : Hardware environment requirements for ESP32-S2 firmware flashing
- Chinese blog : ESP32-S2 USB & UART download summary

Official Documentation:

- ESP32-S2 Datasheet
- ESP32-S2 Hardware Design Guidelines

ESP32-S3

The ESP32-S3 supports two ways to download the firmware: UART0 and USB.

Hardware Connection When you are use the UART0 pins to download the firmware ,the following wiring conditions need to be satisfied:

VDD	->	3V3								
GND	->	GND								
EN	->	Pull	Up	(Used	for	power	on,	can't	be	float)
GPIOO	->	Pull	Down	(Enter	dov	wnload	mode	∋)		
GPIO46	->	Pull	Down							
TXD0(GP)	043	3)	-> RX							
RXD0 (GP)	044	1)	-> TX							

When you are use the USB pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND (shared ground with the power supply board)
EN -> Pull high (used for power-on startup, cannot be left floating)
GPI00 -> Pull low (to enter download mode)
GPI046 -> Pull low
GPI019 -> USB_D-
GPI020 -> USB_D+
```

Note: Strapping pin GPIO45 is used to select the VDD_SPI Flash voltage.

- When using 1.8V VDD_SPI Flash, GPI045 must be pulled high during chip power-up.
- When using 3.3V VDD_SPI Flash, GPI045 must be pulled low during chip power-up.

Startup Conditions

- The operating voltage range for the ESP32-S3 chip is $3.0 \ V \sim 3.6 \ V$. When using a single power supply, it is recommended to use a supply voltage of $3.3 \ V$ with an output current of $500 \ mA$ or higher.
- The operating voltage range for the ESP32-S3 module is $3.0 \ V \sim 3.6 \ V$. When using a single power supply, it is recommended to use a supply voltage of $3.3 \ V$ with an output current of $500 \ mA$ or higher.

References Hardware wiring principle summary:

- · Chinese Blog : Hardware environment requirements for ESP32-S3 firmware flashing
- · Chinese Blog : ESP32-S3 USB & UART download summary

Official documentation:

- ESP32-S3 Datasheet
- ESP32-S3 Hardware Design Guidelines

ESP32-C2

The ESP32-C2 downloads firmware via UARTO (TXD (GPI020) and RXD (GPI019)) by default.

Hardware Connection The following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on Control, can't be float)
GPI08 -> Pull Up (Default is float)
GPI09 -> Pull Down (Default is high level)
TXD0(GPI020) -> RX
RXD0(GPI019) -> TX
```

```
Note:
```

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UARTO serial port.
- The strapping combination of GPIO8 = 0 and GPIO9 = 0 is invalid and will trigger unexpected behavior.

Hardware conditions must be met

- ESP32-C2 chip working voltage range is $3.0 \text{ V} \sim 3.6 \text{ V}$; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.
- ESP32-C2 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.

Reference Summary of Hardware Wiring Principles:

• The hardware conditions for the ESP32-C2 chip to enter the "Download Mode"

Official Documentation:

- ESP32-C2 Datasheet
- ESP32-C2 Hardware Design Guidelines

ESP32-C3

The ESP32-C3 supports two ways to download the firmware: UART0 and USB.

Hardware Connection When you are use the UART0 pins to download the firmware ,the following wiring conditions need to be satisfied:

(Used for power on Control, can't be float)
(Controls the SPI Flash startup mode)
n (Enter download mode)
X
X

When you are use the USB pins to download the firmware ,the following wiring conditions need to be satisfied:

VDD	->	3V3	
GND	->	GND	
EN	->	Pull Up	(Used for power on Control, can't be float)
GPIO2	->	Pull Up	(Controls the SPI Flash startup mode)
GPIO8	->	Pull Up	
GPIO9	->	Pull Down	(Enter download mode)
GPIO18	->	USB_D-	
GPIO19	->	USB_D+	

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UARTO serial port.
- The strapping combination of GPIO8 = 0 and GPIO9 = 0 is invalid and will trigger unexpected behavior.

Hardware conditions must be met

- ESP32-C3 chip working voltage range is $3.0 \text{ V} \sim 3.6 \text{ V}$; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.
- ESP32-C3 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.

Reference Summary of Hardware Wiring Principles:

- Chinese Blog : The hardware conditions for the ESP32-C3 chip to enter the "Download Mode"
- Chinese Blog : ESP32C3 USB & UART Download Mode

官方文档:

- ESP32-C3 Datasheet
- ESP32-C3 Hardware Design Guidelines

ESP32-C5

ESP32-C5 supports two firmware download methods: UART0 and USB.

Hardware Wiring When using the UARTO pin for firmware download, it supports two boot modes: Joint Download Boot 0 and Joint Download Boot 1. Different boot modes have different default level requirements for the Straping pin, as detailed below:

• In Joint Download Boot 0 mode, the following wiring conditions need to be met:

```
VDD -> 3V3
GND -> GND (Common ground with the power board)
EN -> Pull high (Used for power-on startup, cannot be left floating)
GPI027 -> Pull high (Default weak pull-up)
GPI028 -> Pull low (Default is ``high level``)
TXD0(GPI011) -> RX
RXD0(GPI012) -> TX
```

• In Joint Download Boot 1 mode, the following wiring conditions need to be met:

```
VDD
       -> 3V3
       -> GND
GND
                (Common ground with the power board)
       -> Pull high (Used for power-on startup, cannot be left floating)
EN
        -> Pull low (Default is floating)
GPIO26
        -> Pull Low (Default weak Pull-up)
GPT027
       -> Pull low (Default is ``high level``)
GPT028
TXD0(GPI011)
              -> RX
RXD0(GPI012)
               -> TX
```

- In addition, EFUSE_XTAL_48M_SEL_MODE needs to be 0;
- And GPIO2 (i.e., MTMS) should be set to high level (48MHz) or low level (40MHz) according to the size of the crystal used by the chip.

If EFUSE_XTAL_48M_SEL_MODE is 1, when using a *40MHz* crystal, EFUSE_XTAL_48M_SEL(0b000) should have an *even number of 1s*, and the level of GPIO2 (i.e., MTMS) is ignored. If EFUSE_XTAL_48M_SEL_MODE is 1, when using a *48MHz* crystal, EFUSE_XTAL_48M_SEL(0b000) should have an *odd number of 1s*, and the level of GPIO2 (i.e., MTMS) is ignored.

When using the USB pin for firmware download, only Joint Download Boot 0 mode is supported, and the following wiring conditions need to be met:

VDD -> 3V3
GND -> GND (Common ground with the power board)
EN -> Pull high (Used for power-on startup, cannot be left floating)

(continues on next page)

(continued from previous page)

```
GPI027 -> Pull high (Default weak pull-up)
GPI028 -> Pull low (Default is ``high level``)
GPI013 -> USB_D-
GPI014 -> USB_D+
```

Note:

- After powering on the chip/module, you can check whether it has entered the Download Boot mode through the UARTO serial port.
- In Joint Download Boot 0 mode, when the chip is powered on, GPI027 and GPI028 cannot both be at low level.

Boot Conditions

- The working voltage range of the ESP32-C5 chip is $3.0 \text{ V} \sim 3.6 \text{ V}$; when using a single power supply, it is recommended that the power supply voltage for the ESP32-C5 series chip be 3.3 V, and the rated output current should preferably be 800 mA or above.
- The operating voltage range of the ESP32-C5 module is $3.0 \ V \sim 3.6 \ V$; when using a single power supply, it is recommended that the power supply voltage for the ESP32-C5 series chip be $3.3 \ V$, and the rated output current should ideally be $800 \ mA$ or above.

When testing with the ESP32-C5-DevKitC-1 development board, you can directly use a USB Type-C cable to connect the UART or USB interface on the development board for direct firmware download. If you are using the USB interface to download the firmware for the first time, you need to manually pull down the GPIO28 pin, that is, hold down the Boot button and then power on, to manually enter the download mode.

ESP32-C6

The ESP32-C6 supports two ways to download the firmware: UART0 and USB.

Hardware Connection When you are use the UART0 pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on Control, can't be float)
GPI08 -> Pull Up (Default is float)
GPI09 -> Pull Down (Default is high level)
TXD0 (GPI016) -> RX
RXD0 (GPI017) -> TX
```

When you are use the USB pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on Control, can't be float)
GPI08 -> Pull Up (Default is float)
GPI09 -> Pull Down (Default is high level)
GPI012 -> USB_D-
GPI013 -> USB_D+
```

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UARTO serial port.
- The strapping combination of GPIO8 = 0 and GPIO9 = 0 is invalid and will trigger unexpected behavior.

Hardware conditions must be met

- ESP32-C6 chip working voltage range is $3.0 \ V \sim 3.6 \ V$; if you use a single power supply, the recommended voltage of the power supply is $3.3 \ V$, and its recommended output current is $500 \ mA$ or more.
- ESP32-C6 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 500 mA or more.

Reference Summary of Hardware Wiring Principles:

• Chinese Blog : The hardware conditions for the ESP32-C6 chip to enter the "Download Mode"

Official Documentation:

- ESP32-C6 Datasheet
- ESP32-C6 Hardware Design Guidelines

ESP32-H2

The ESP32-H2 supports two ways to download the firmware: UART0 and USB.

Hardware Connection When you are use the UART0 pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on Control, can't be float)
GPI08 -> Pull Up (Default is float)
GPI09 -> Pull Down (Default is high level)
TXD0(GPI024) -> RX
RXD0(GPI023) -> TX
```

When you are use the USB pins to download the firmware ,the following wiring conditions need to be satisfied:

```
VDD -> 3V3
GND -> GND
EN -> Pull Up (Used for power on Control, can't be float)
GPI08 -> Pull Up (Default is float)
GPI09 -> Pull Down (Default is high level)
GPI026 -> USB_D-
GPI027 -> USB_D+
```

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UARTO serial port.
- The strapping combination of GPIO8 = 0 and GPIO9 = 0 is invalid and will trigger unexpected behavior.

Hardware conditions must be met

- ESP32-H2 chip working voltage range is $3.0 \text{ V} \sim 3.6 \text{ V}$; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 350 mA or more.
- ESP32-H2 modules working voltage range is 3.0 V ~ 3.6 V; if you use a single power supply, the recommended voltage of the power supply is 3.3 V, and its recommended output current is 350 mA or more.

Reference Summary of Hardware Wiring Principles:

• Chinese Blog : The hardware conditions for the ESP32-H2 chip to enter the "Download Mode"

Official Documentation:

- ESP32-H2 Datasheet
- ESP32-H2 Hardware Design Guidelines

ESP32-P4

ESP32-P4 supports two firmware download methods: UART0 and USB. Different boot defaults require different levels for the Straping pin, as detailed below:

Hardware Wiring When downloading firmware using the UARTO pin, the following wiring conditions need to be met:

```
VDD -> 3V3
GND -> GND (Common ground with the power board)
EN -> Pull high (Used for power-on startup, cannot be left.
→floating)
GPI036 -> Pull high (Default floating)
GPI035 -> Pull low (Default is ``high level``)
TXD0(GPI037) -> RX
RXD0(GPI038) -> TX
```

When downloading firmware using the USB pin, the following wiring conditions need to be met:

```
VDD -> 3V3
GND -> GND (Common ground with the power board)
EN -> Pull high (Used for power-on startup, cannot be left floating)
GPI036 -> Pull high (Default floating)
GPI035 -> Pull low (Default is ``high level``)
GPI024(GPI026) -> USB_D-
GPI025(GPI027) -> USB_D+
```

Note:

• After powering on the chip/module, you can check whether it has entered the Download Boot mode through the UARTO serial port. If the chip is powered on and enters the download mode, UARTO will print the following log:

```
ESP-ROM:esp32p4-eco1-20240205
Build:Feb 5 2024
rst:0x1 (POWERON),boot:0x307 (DOWNLOAD(USB/UART0/SPI))
waiting for download
```

Startup Conditions

• The working voltage range of the ESP32-P4 chip is $3.0 \ V \sim 3.6 \ V$; when using a single power supply, it is recommended that the power supply voltage for the ESP32-P4 series chip is $3.3 \ V$, and the rated output current is preferably $500 \ mA$ or above.

When testing with the ESP32-P4-Function-EV-Board development board, you can directly use the USB Type-C cable to connect the USB-UART interface on the development board to download the firmware directly. If you are using the USB interface to download the firmware for the first time, you need to manually pull the GPIO35 pin low, that is, press and hold the Boot button and then power on, to manually enter the download mode.

Selecting the Suitable Flashing Platform

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Espressif officially provides the following flashing methods for different needs:

- 1. Quick Firmware Testing: ESP LAUNCHPAD Quick Firmware Flashing
 - Advantage: Simple and fast, no need to set up an environment or additional tools, firmware can be easily flashed through the Web interface
- 2. Non-developers performing flashing: Flash Download Tool Flashing
 - Advantage: Provides a graphical interface, no need to understand command line operations in depth, also suitable for automated deployment and batch production flashing
- 3. Professional developers for comprehensive firmware development: IDE Flashing
 - Advantage: Integrated Development Environment (IDE) provides an intuitive user interface and toolset, supports firmware development, debugging and integrated flashing
- 4. **Professional developers** for comprehensive **firmware development and more detailed configuration:** *IDF Terminal Flashing*
 - Advantage: Allows developers to precisely manage projects, including firmware parameter settings. Convenient for flashing during development and debugging to quickly verify changes

Preparation



Note: If you haven't selected a suitable development board for your project yet, please refer to *Board* selection

For the differences between chips, modules, and development boards, please refer to *Chips, Modules, Development Boards*

Using the Development Board You can flash by connecting the USB or UART interface with a data cable.

Note: Some development boards use the USB Type C interface. Please make sure to use the appropriate data cable.

Using the Chip or Module Espressif chips generally have two modes:

- 1. Firmware Flashing Mode: The chip will wait to receive firmware from the serial port for flashing. Usually, a flashing tool is needed to send firmware data
- 2. Normal Operating Mode: This is the normal working mode of the chip, it will load and run the firmware from the Flash storage

To use the chip or module, you need to enter the flashing mode by setting the status of **specific pins**, and flash through the **UART0** or **USB** peripheral.

Note: Specific pin information should be referenced from the corresponding chip's technical datasheet, which can be found on the Espressif official website.

ESP LAUNCHPAD Quick Firmware Flashing ESP LAUNCHPAD is a platform that allows developers to quickly experience the features of Espressif chips.

- After selecting some example firmware provided by Espressif, you can directly flash it on the web page
- Developers can also upload their compiled . bin files and flash them on the web page

ESP LAUNCHPAD Flashing Process The following uses ESP-BOX as an example:

Open the website ESP LAUCHPAD

- 1. Click on the top right **Connect**
- 2. In the pop-up window, select the **Port** to connect to the ESP-BOX
- 3. Click **Connect** below

connect seron				
espressif.github.io 想连接到串行满口 CP2102N USB to UART Bridge Controller (COM 19) 2	PAD	€ Quick Start	DIY 1 Connect	Console Settings About
	ash the selected firmware image onto y our device to the serial USB port. Click or vice, any earlier firmware would be oven	rour device. n the 'Connect' button in the to written.	p menu option, to conne	act to your attached device.
© 3	e images listed below. e firmware images from an <u>exte</u>	e <u>rnal source</u> - https://ray	w.githubuserconter	it.com/espressif/esp-
Select Application ESP Chipset Type	ESP-BOX_DEMO_CN_V0_5_C	0 -		

4. Select the firmware to be flashed to the ESP-BOX
| Select Application | ESP-BOX_Demo_CN_V0_5_0 |
|--------------------|--------------------------------|
| | ESP-BOX_Demo_EN_V0_5_0 |
| ESP Chipset Type | ESP-BOX_Demo_CN_V0_5_0 |
| | ESP-BOX-Lite_Demo_EN_V0_5_0 |
| | ESP-BOX-Lite_Demo_CN_V0_5_0 |
| | ESP-BOX_ChatGPT_Demo_V1_0 |
| | ESP-BOX-Lite_ChatGPT_Demo_V1_0 |
| | ESP-BOX_USB_Headset_V0_0_1 |
| | ESP-BOX_Demo_EN_V0_3_0 |
| | ESP-BOX_Demo_CN_V0_3_0 |

- 5. Select ESP Chipset Type as ESP32-S3
- 6. Click the bottom left Flash button to start flashing
- 7. After the flashing starts, it will jump to the console interface, if correct, the flashing progress will be displayed

Connected to devi	ce: ESP32-S3	
ESP Launchpad hel	ps you to flash the selected firmware image onto your device.	
writing at 0x10046. writing at 0x1126cf. Writing at 0x1124cf. Writing at 0x124cf. Writing at 0x124cf. Writing at 0x124cf. Writing at 0x124cf. Writing at 0x124cf. Writing at 0x1405f. Writing at 0x1405f. Writing at 0x156f. Writing at 0x166f. Writing at 0x164f. Writing at 0x164f.	(58) (58) (58) (53) (53) (53) (53) (63) (63) (63) (63) (75) 	

8. After the flashing is completed, press the top left **Reset Device** button, or click **Disconnect** in the top right corner.

Flash Download Tool Flashing The Flash Download Tool is a tool developed by Espressif for flashing firmware to its ESP series chips, used to load pre-compiled firmware files into the chip's flash memory, enabling the chip to correctly run applications or firmware.

 Download Flash Download Tool Refer to Flash Download Tool User Guide for more information.

Using the Flash Download Tool Flashing Process

- 1. Run the .exe file of flash_download_tool
- 2. Select **Development Board (ChipType)** as the corresponding development board, there will generally be a mark on the chip. Select **Flashing Mode (LoadMode)** as UART and then click **OK** below

C:\Users\yanpengyu\Desktop\flash_download_tool_3.9.5\flash_d	ownload	_tool_3.9	.5.exe				-	\times
E D	W	-		×				
Chip	Type:	ESP32-S	\$3	\sim				
		D						
Wor	kMode:	Develop	Þ	~				
Load	Mode:	UART] ~				
		OK						

- 3. Click to select the .bin file to be flashed
- 4. Enter the **address offset** to be flashed, you can default to the starting position 0×0
- 5. Select the **COM port** connected to the development board
- 6. Click **START** at the bottom left to start flashing

ESP32S3 F		ILOAD TOC	DL V3.9.5		_		\times
SPIDownloa	d						
Ownload 3 3 1	SPI MODE QIO QUUT OQUUT ODUUT FASTRD		/0.3.0.bir NotChgBi Settings mbineBir Default			0x0 4 0x0 0x0 0x0 0x0 0x0 0x0 0x	
DownloadPan	iel 1						
IDLE							
等待			5				~
START	STOP	ERASE	COM:	CON	M15		~
			BAUD:	115	200		\sim

7. The interface will display a blue FINISH after the flashing is completed



Use Flash Download Tool for Flash encryption and secure boot

- 1. Disable all secure boot and Flash encryption configurations in **menuconfig** to ensure that plaintext firmware can be generated after compilation
- 2. Adjust the offset of the default partition table (Offset of partition table), adjust from 0x8000 to 0xa000

Note: Flash encryption will increase the size of the bootloader .bin firmware, so you must leave

enough space for the offset address

3. Open the folder of **Flash Download Tool**, open **configure** -> **esp chip type** -> **security.conf**. Enable the following configurations and save the file:

```
[SECURE BOOT]
secure_boot_en = True
[FLASH ENCRYPTION]
flash_encryption_en = True
reserved_burn_times = 3
[ENCRYPTION KEYS SAVE]
keys_save_enable = True
encrypt_keys_enable = False
encrypt_keys_aeskey_path =
[DISABLE FUNC]
jtag_disable = False
dl_encrypt_disable = False
dl_decrypt_disable = False
dl_cache_disable = False
```

- 4. Restart Flash Download Tool
- 5. Add the generated plaintext firmware to the Flash Download Tool and set the corresponding offset address
- 6. Use the Flash Download Tool to flash the .bin plaintext firmware according to the above *Using the Flash Download Tool Flashing Process*, the encryption key will be saved locally

flash download tool 3.9.2 0 > flash download tool 3.9.2 > secure >

-		-
* ^	名称	修改日期
*	encrypt_keys.csv	2022/9/1 19:27
*	T flash_encrypt_key_1.bin	2022/9/1 19:27
*	flash_encrypt_key_1.pem	2022/9/1 19:27
	secure_boot_key_1.bin	2022/9/1 19:27
	secure_boot_key_1.pem	2022/9/1 19:27

IDE Flashing After setting up the development environment, you can use the ESP-IDF plugin in the IDE to build and flash.

• Choose an instance project of interest from esp-iot-solution or the built-in examples of ESP-IDF.

Note: For environment setup, refer to: VSCode ESP IDF Extension

We can experience the flashing process of ESP32-S3-BOX, this project uses ESP32-S3-BOX as a USB speaker.

Take VSCode as an Example:

- 1. Enter the path where you want to save the project in VSCode, and select **Terminal** -> **New Terminal** in the upper left window
- 2. Use git clone to download the code of this project

git clone --recursive **https:**//github.com/espressif/esp-box.git

- 3. Open VSCode, select **file** -> **open folder** (or use the shortcut Ctrl + K, Ctrl + O) in the upper left window to enter the path where the project is saved -> **esp-box** -> **examples** -> **usb_headset**. Click to select the folder.
- 4. Connect the ESP-BOX to the computer via a data cable

ξ ³ 25 > TIMELINE 6			7		8				
> PROJECT COMPONEN	ITS		<u>'</u>		<u> </u>				
♥ COM1 🛛 🖾 esp32s3 🖻	۵ (Û	٥	★	в	ا	▷	€	⊗ 0 ∆ 0

5. Select the corresponding COM port in the lower left corner

Note: If the COM port is not detected, hold down the **Boot** key and the **Reset** key of the development board at the same time to enter the download mode

- 6. Select Target as esp32s3, VSCode selects ESP32-S3 chip (via ESP-PROG) at the top of the interface
- 7. Click the ESP-IDF Build Project icon, after successful compilation, it will display:

Total sizes:								
Used stat D/IRAM:	91594	bytes	(25426	2 re	main,	26.5%	used)
.data size:	11392	bytes						
.bss size:	5408	bytes						
.text size:	73767	bytes						
<pre>.vectors size:</pre>	1027	bytes						
Used Flash size :	290627	bytes						
.text :	216459	bytes						
.rodata :	73912	bytes						
Total image size:	376813	bytes	(.	bin m	ay b	e pado	ded la	rger)
Ĭ								

- 8. Click the **ESP-IDF Flash device** icon to flash, then select **UART** at the top of the VSCode interface. After successful flashing, it will display in the terminal: Flash Done *5*
- 9. After pressing the **Reset** button on the development board, you can use the ESP-BOX to play sound via USB

IDF Terminal Flashing Using the IDF terminal has advantages such as stable environment, easy version switching, and full advanced features.

Note: For environment setup, you can refer to: Windows Installation Guide.

The following will introduce how to use the IDF terminal to flash an example project:

1. Open the ESP-IDF CMD terminal window, the interface of successful installation of ESP-IDF SDK is shown as follows:



3. Switch to the correct chip environment, for example:

```
idf.py set-target esp32
```

Note: Replace esp32 with the correct chip target

4. Run the following command to build the project:

idf.py build

5. Run the following command to flash the project, replace PORT with the serial name of the development board:

```
idf.py -p PORT flash
```

Note: Using idf.py flash will attempt to automatically connect using the available serial port

The interface of successful flashing is shown as follows:



Obtaining Firmware Flashing Information for Different Software Development Platforms (Development Stage)

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

The firmware of a complete project developed based on the ESP-IDF SDK usually includes bootloader.bin, partition

- bootloader.bin: is the second-stage bootloader
- partition-table.bin: is the firmware for managing the Flash partition table, for detailed explanation, see: Partition Table
- app.bin: is the firmware for executing user application functions

The download address (also called offset address) of the bootloader.bin of the non-series chip is fixed and does not support modification. The specifics are as follows:

Chip	bootloader.bin Download Address
ESP8266/ESP8285	0x0
ESP32	0x1000
ESP32-C2/ESP8684	0x0
ESP32-C3/ESP8685	0x0
ESP32-C5	0x2000
ESP32-C6	0x0
ESP32-S2	0x0
ESP32-S3	0x0
ESP32-H2	0x0
ESP32-P4	0x0

 Table 2: Chip and bootloader.bin Download Address

The specific firmware and corresponding download address generated by the project depend on the project's partition table settings. Each project needs to set the corresponding partition table. The ESP-IDF SDK provides a series of default partition table files for users to use, see: esp-idf/components/partition_table in the .csv files.



Users can also set custom partition table files for the project, refer to: esp-idf/examples/storage/spiffs example. Create

a .csv file in the project directory, and then refer to the Partition Table usage instructions for settings, such as the partition table file below:

esp-idf / e	xamples / storage / spiffs / partitions_example.csv 🖓
We can n	nake this file beautiful and searchable if this error is corrected: It looks like row 2 should actually have 6 columns, instead o
😥 pro	jectgus and espressif-bot Whitespace: Automated whitespace fixes (large commit)
Preview	Code Blame 6 lines (6 loc) · 311 Bytes
1	# Name, Type, SubType, Offset, Size, Flags
1 2	# Name, Type, SubType, Offset, Size, Flags # Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap
1 2 3	# Name, Type, SubType, Offset, Size, Flags # Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap nvs, data, nvs, 0x9000, 0x6000,
1 2 3 4	# Name, Type, SubType, Offset, Size, Flags # Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap nvs, data, nvs, 0x9000, 0x6000, phy_init, data, phy, 0xf000, 0x1000,
1 2 3 4 5	# Name, Type, SubType, Offset, Size, Flags # Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap nvs, data, nvs, 0x9000, 0x6000, phy_init, data, phy, 0xf000, 0x1000, factory, app, factory, 0x10000, 1M,

If the project uses a custom partition table file, you will need to set the custom partition table name and path in the project software configuration, as follows:



The software development of ESP series products supports various software development environments, commonly used software environments include ESP-IDF CMD terminal compilation environment, VSCODE IDE, Arduino IDE and other software environments.

- For the ESP-IDF CMD terminal compilation environment under the Windows environment, specific usage instructions can be referred to: [In Chinese]Setting up ESP-IDF SDK Compilation Environment + Visual Studio Code Software Programming Environment on Windows
- VS CODE IDE software development environment, specific usage instructions can be referred to: [In Chinese]ESP-IDF Extension for VS Code
- Arduino IDE software development environment, for specific usage instructions, refer to: [In Chinese]Installing ESP32 Arduino Software Development Environment on Windows

1. Windows ESP-IDF CMD Software Compilation Environment In the Windows ESP-IDF CMD software compilation environment, project configuration, compilation, download, and monitoring are all completed by executing specific commands. For example:

Users can execute the idf.py menuconfig command for project configuration

Partition table settings:



Partition table download address setting:



After compiling the project using the Windows ESP-IDF CMD software compilation environment, the following log will be printed after the idf.py build command is executed to display the firmware generated by the current project and the corresponding firmware download address.



Users can also directly open the flash_project_args file in the build folder to obtain firmware download information, as follows:

t (E:) > esp	> Espressif >	frameworks > esp-idf-v5.0.2 >	examples > get-started	> hello_world > build >	C)	≣ flash_project_args ×	ESP-IDF: 搜	索错误提示	
	ENV 文件 6.62 KB		文件 77 字节	^		E: > esp > Espressif > frar 1flash_mode	meworks >esp-idf-v5.0.2 >examples >get-started >hello_world >build > ⊫ flash_ e dioflash_freq 80mflash_size 2MB	project_arg	s
	flash_args 文件 152 字节		flash_args.in IN 文件 192 字节	/	re So	2 0x0 bootload 3 0x10000 hell 4 0x8000 parti	ler/bootloader.bin .o_world.bin .tion_table/partition-table.bin		
	flash_boot 文件 83 字节	loader_args	flash_project_args 文件 152 字节		da B				

Finally, users can directly run the idf.py flash monitor command to download the firmware and monitor the firmware operation log.

After obtaining the firmware and address compiled by the project, you can also refer to the *Selecting the Suitable Flashing Platform* section for firmware download.

2. VSCODE IDE Software Compilation Environment In the VS CODE IDE environment, shortcut buttons are provided for software compilation, flashing, configuration, monitoring, and debugging, and users can directly complete specific settings through these buttons. For example:

Software settings: equivalent to the idf.py menuconfig configuration interface.

文件(F) 编辑(E) 选择(S) 3	查看(V) 转到(G) ··· ← →	, P station	· · · · · · · · · · · · · · · · · · ·
资源管理器	툴 flash_project_args ● ESP-IDF 欢迎 C sta	tation_example_main.c 9+ ● SDK配置编辑器 ×	
> 打开的编辑器			
I I I I I I I I I I I I I I I I I I I			保存
🍿 ESP-IDF 欢迎			
C station_ex 9+	Build type	Partition lable	
× 🏶 SDK配置编辑器	 Bootloader config 	Partition Table 🛈	
✓ STATION	Bootloader manager		
> .vscode	Serial Flash Configurations	Single factory app, no OTA 分区表述	off ➤
> build	Security features	Offset of partition table ()	
🗸 🗸 🗸 🗸 🗸	Application manager		
M CMakeLists.txt	Social flacker config	0x8000	
≡ Kconfig.projbuild	Partition Table 公区丰公署部署	Concerts on MDE aboutions for the new	
C station_exam 9+	Example Configuration		
M CMakeLists.txt	 Compiler options 	Example Configuration	
③ README.md	Replace ESP-IDF and project paths in binaries	WiFi SSID ①	
sdkconfig	Enable C++ exceptions		
sdkconfig.ci	 Component config 	myssid	
sdkconfig.ci.esp32c2	Application Level Tracing	WiFi Password	
sdkconfig.defaults	Bluetooth Consolo Library		
sdkconfig.old	 Driver Configurations 	mypassword	
	TWAI Configuration	WPA3 SAE mode selection ①	
	 Legacy ADC Driver Configuration 		
	Legacy ADC Calibration Configuration		
	Legacy MCPWM Driver Configurations		
)	Legacy Timer Group Driver Configurations	PASSWORD IDENTIFIER U	
	Legacy RMT Driver Configurations		
)	Legacy PCNT Driver Configurations		
	Legacy SDM Driver Configurations	Maximum retry 🕕	
	Legacy Temperature Sensor Driver Configurations		
	eFuse Bit Manager	WEELCoop with my do thread and O	
	ADC and ADC Calibration	WiFi Scan auth mode threshold ()	
	Wireless Coexistence	WPA2 PSK	
	Common ESP-related		
	ESP-Driver:GPIO Configurations	Compiler options	
	ESP-Driver:GPTimer Configurations	Optimization Level 🕕	
	ESP-Driver: I2C Configurations		
	ESP-Driver:I2S Configurations	Debug (-Og) 🗸 🗸	
	ESP-Driver:LEDC Configurations	Assertion level	
	ESP-Driver:PCNT Configurations	Assertion level	
〉大纲	ESP-Driver:RMT Configurations	Enabled ×	
〉时问线	monor of one state of		
> 项目组件	menuconfig 设置		
0 0 0 0 0 0			

Project Flashing: equivalent to idf.py flash.

During the project Flashing process, the firmware and corresponding download address of the project flashing will also be printed on the terminal running interface, as follows:

终端 阔口 GITLENS ESP-IDF 调试控制台问题 112 输出	
Merged 2 ELF sections	≥ powershell
Successfully created esp32s3 image.	上 ESP-IDF Flash 任务
Generated E:/esp2/Espressif/frameworks/esp-idf-master/esp-idf/examples/wifi/getting_started/station/build/wifi_station.bin [1004/10004] cmd.exe /C "cd /D E:\esp2\Espressif\frameworks\esexamples/wifi/getting_started/station/build/wifi_station.bin" wifi_station.bin binary size 0xb0b90 bytes. Smallest app partition is 0x1000000 bytes. 0x4f470 bytes (31%) free.	上 反 python 任务
I 正在执行任务: e:\esp-idf-vscode\python_env\idf5.3_py3.11_env\Scripts\python.exe e:\esp\Espressif\frameworks\v5.3\esp-idf\c omponents\esptool_py\esptool\esptool.py -p COM4 -b <u>dF60800before default_resetafter hard_resetafter hard_res</u>	
Meryal Critin Caule.com	
esptol.py V4.8.0 Serial port COM Connecting Chip is ESP22-53 (QFM56) (revision V0.1) Features: WiFi, BLE, Embedded PSRAM 2MB (AP_3V3) Crystal is 40MHz MAC: 68:b6:b3:55:41:74 Uploading stub Running stub Running stub Running Changing baud rate to 460800 Changed. Configuring flash size Flash will be erased from 0x00000000 to 0x00005fff Flash will be erased from 0x00000000 to 0x00005fff Flash will be erased from 0x00000000 to 0x00005fff Flash will be erased from 0x00000000 to 0x000005fff Flash will be erased from 0x00000000 to 0x000000 fff Flash will be erased from 0x00000000 to 0x0000000 to 0x0000000000	
Compressed 75050 bytes to 45051 Writing at 0x00087028 (71 %)[] 烧家固件	
iESP-IDF v5.3.0 ★ UART Ϋ COM4 🗘 esp32s3 日 🎒 🤌 🤌 💭 🎝 🔽 🗗 🛞 🗵 🕑 🛞 12 🛆 0 🎉 1 👷 0 日 😂 生成 🔅 ▷ 🔿 ESP-IDF: Flashi	ng project

After the project burning is completed, the log of the command running during the firmware flashing will also be printed, as follows:



Similarly, you can also directly open the *flash_project_args* file in the *build* folder generated by the project compilation to obtain firmware download information, as follows:



After obtaining the firmware and address of the project compilation, you can also refer to the instructions in the *Selecting the Suitable Flashing Platform* section for firmware download.

3. Arduino IDE Software Compilation Environment

Open the example in Arduino IDE:

	DE 2.3.2	Built-in examples	
le Edit Sketch	Ioois Heip	01.Basics	•
New Sketch	Ctrl+N	02.Digital	
	ctul o	03.Analog	
Open	Ctri+O	04.Communication	
Open Recent	P .	05.Control	
Sketchbook	P	06.Sensors	
Examples	• • • • • • • • • • • • • • • • • • •	07.Display	► dlv.
Close	Ctrl+w	08.Strings	
Save	Ctri+S	09.USB	MEGA and ZERO
Save As	Ctri+Snift+S	10.StarterKit_BasicKit	► IN is set to
Preferences	Ctrl+逗号	11.ArduinoISP	
Advanced	•	Examples for ESP32S3 Dev Module	your Arduino
Quit	Ctrl+O	ArduinoOTA	▶
	currq	BLE	Beacon_Scanner
13	modified 8 May	BluetoothSerial	BLE5_extended_scan
14	by Scott Fitzg	DNSServer	BLE5_multi_advertising
15	modified 2 Sep	EEPROM	BLE5_periodic_advertising
10	modified & Sen	ESP Insights	BLE5_periodic_sync
18	by Colby Newma	ESP RainMaker	Client
19		ESP_I2S	EddystoneTLM_Beacon
20	This example c	ESP_NOW	EddystoneURL_Beacon
21		ESP_SR	▶ iBeacon
22	https://www.ar	ESP32	Notify
23	*/	ESP32 Async UDP	► Scan
24	// the setup fun	ESPmDNS	► Server
26	<pre>void setup() {</pre>	Ethernet	Server_multiconnect
27	// initialize	FFat	► UART
28	<pre>pinMode(LED_BU</pre>	HTTPClient	► Write
29	}	HTTPUpdate	•
30	// the less Core	HTTPUpdateServer	•
31	void loon() {	LittleFS	▶
33	digitalWrite(L	Matter	<pre>e voltage level)</pre>
34	delay(1000);	NetBIOS	•
35	digitalWrite(L	NetworkClientSecure	the voltage LOW
36	delay(1000);	OpenThread	•
37	}	РРР	•
38		Preferences	•
		SD	▶
		SD_MMC	▶
		SimpleBLE	▶
		SPI	▶
		SPIFFS	•
		TFLite Micro	▶
		Ticker	▶
		Update	►
		USB	►
		WebServer	►
X)		WiE;	

Arduino IDE Software Configuration:

☑ I2S_T	oggle_Test 4	Arduino IDE 2.3.2				сн 🖉 🔁 🗧
File Ed	it Sketch To	ools Help		e. 1	-	
	⇒ ₽	Auto Format		Ctrl+	1	
-	BOARDSIN	Manage Libraries		Ctrl+Shift		
		Serial Monitor		Ctrl+Shift+	u l	
	esp32	Serial Plotter		curromiter	"	
白	Туре:					
		Firmware Updater				
MA	Arduinc	Upload SSL Root Ce	rtificates			
	2 0 18 an	Board: "ESP32S3 De	v Module"			
	Roards in	Port: "COM4"				
÷.	Arduino N	Get Board Info				
	More info	USB CDC On Boot: "	Disabled"			
Q	2.0.18-11	CPU Frequency: "24	OMHz (WiFi)	•	•	
		Core Debug Level: *	None"		•	
		USB DFU On Boot: "	Disabled"			
	esn32 b	Erase All Flash Befor	e Sketch Up	load: "Disabled"		
	Systems	Events Run On: "Cor	'e 1"			
	3.1.0 inst	Flash Mode: "QIO 80	JMHz"			
	Boards in	Flash Size: "4IVIB (32	MD)"			1g, 0, NULL);
	Adafruit F	Arduino Runs On: "(Ore 1"			
	More info	LISB Firmware MSC	On Boot: "D	sabled"		
	310	Partition Scheme: "E	efault 4MB	with spiffs (1.2MB APP/1.5MB SPIFFS)"	•	✓ Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)
	0.110	PSRAM: "Disabled"			Þ	Default 4MB with ffat (1.2MB APP/1.5MB FATFS)
		Upload Mode: "UAR	T0 / Hardwa	re CDC"	•	8M with spiffs (3MB APP/1.5MB SPIFFS)
		Upload Speed: "921	600"		•	Minimal (1.3MB APP/700KB SPIFFS)
		USB Mode: "Hardwa	ire CDC and	JTAG"	•	No FS 4MB (2MB APP x2)
		Zigbee Mode: "Disa	bled"		►	No OTA (2MB APP/2MB SPIFFS)
		Programmer			•	No OTA (1MB APP/3MB SPIFFS)
		Burn Bootloader				No OTA (2MB APP/2MB FATFS)
	_		65			No OTA (1MB APP/3MB FATFS)
			66	(Lucit o consta for UCD to be puit		Huge APP (3MB No OTA/1MB SPIFFS)
			67	// wall 2 seconds for USB to be avai.	TODT	MINIMAL SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
			69			TOM Flash (2MB APP/12.5MB FATES)
			70	<pre>nextToggle = millis() + 2000;</pre>		Tow Flash (SWB APP/9.9MB FATES)
	Output					RainMaker 4MB No OTA
						RainMaker 8MB
			no r	32M Flash (4.8MB APP/22MB FATFS)		
			Runnin	g pre_uninstall script.		32M Flash (4.8MB APP/22MB LittleFS)
			Tool e	sp32:esptool_py@4.8.1 uninstalled		ESP SR 16M (3MB APP/7MB SPIFFS/2.9MB MODEL)
			Unist	alling esp32:mklittlefs@3.0.0-gnu12-dc7	71933	Zigbee ZCZR 4MB with spiffs
			Uninst	alling esp32:mkspirrs@0.2.3, tool is no alling esp32:openocd-esp32@v0.12.0-esp	32-26	Custom

Arduino IDE Firmware Download Log:

🔤 12S	Toggle_Test Arduino IDE 2.3.2	X
File E	lit Sketch Tools Help	
Ø	🔁 🚱 🦞 ESP32S3 Dev M	odule 🔹
Ph	BOARDS MANAGER	125_Toggle_Testino
	esp32	24 i2s_pin_config_t i2s_mic_pins = {
白	Type: All 🗸	Output
	Advance SEP32 Boards by Advance 2.0.1 Sensing Standard Boards included in the package Advance Standard Company Markowski (Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard Standard PSSAL Www.shaher (SP32-33) PSSAL Www.shaher (SP32-33	espion.py w.g. were manomous bytes to file Clubsers/Laiguanhomg/Applitule.callyremplandmice/late/htms/WE970001110650000201146575000020014657500000000000000000000000000000000000
		Hash of data verified.

Copy the log out, you can get the firmware and download address information corresponding to the project. As follows:

```
esp32s3 --port "COM4" --baud 921600 --before default_reset --after hard_

->reset write_flash -z --flash_mode keep --flash_freq keep --flash_size_

->keep 0x0 "C:\Users\caiguanhong\AppData\Local\Temp\arduino\sketches\

->7B5F00811F0549D692D11436750291AC/I2S_Toggle_Test.ino.bootloader.bin"_

->0x8000 "C:\Users\caiguanhong\AppData\Local\Temp\arduino\sketches\

->7B5F00811F0549D692D11436750291AC/I2S_Toggle_Test.ino.partitions.bin"_

->0xe000 "C:\Users\caiguanhong\AppData\Local\Arduino15\packages\esp32\

->0xe000 "C:\Users\caiguanhong\AppData\Local\Arduino15\packages\esp32\

->oxe000 "C:\Users\caiguanhong\AppData\Local\Arduino15\packages\esp32\

->caiguanhong\AppData\Local\Temp\arduino\sketches\

->7B5F00811F0549D692D11436750291AC/I2S_Toggle_Test.ino.bin" esptool.py v4.

->8.1
```

C:) > 用户 > caiguanhong > AppData	› Local → Temp → arduino → sk	etches → C2D27B83EF	FF52DF6665A3
名称	修改日期	类型	大小
📕 core	2025/1/2 19:55	文件夹	
📙 libraries	2025/1/2 19:55	文件夹	
📕 sketch	2025/1/2 19:55	文件夹	
last-used	2025/1/2 19:55	LAST-USED 文件	0 KB
🕕 build.options.json	2025/1/2 19:55	JSON 源文件	1 KB
🖻 build_opt.h	2025/1/2 19:55	C Header 源文件	0 KB
compile_commands.json	2025/1/2 19:56	JSON 源文件	6 KB
file_opts	2025/1/2 19:55	文件	0 KB
includes.cache	2025/1/2 19:55	CACHE文件	2 KB
libraries.cache	2025/1/2 19:55	CACHE文件	1 KB
partitions.csv	2024/12/31 17:13	XLS Worksheet	1 KB
sdkconfig	2024/12/31 17:13	文件	122 KB
Simple_tone.ino.bin	2025/1/2 19:56	BIN 文件	320 KB
🖹 Simple_tone.ino.bootloader.bi	n 2025/1/2 19:55	BIN 文件	19 KB
Simple_tone.ino.elf	2025/1/2 19:56	ELF 文件	5,938 KB
Simple_tone.ino.map	2025/1/2 19:56	MAP文件	8,939 KB
Simple_tone.ino.merged.bin	2025/1/2 19:56	BIN 文件	4,096 KB
Simple_tone.ino.partitions.bin	2025/1/2 19:56	BIN 文件	3 KB

Then, you can find the firmware of the corresponding project according to the path information provided by the log, as follows:

After obtaining the firmware and address of the project compilation, you can also refer to the *Selecting the Suitable Flashing Platform* chapter for firmware download instructions.

Mass Production Flashing

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

The *Flash Download Tool Flashing* in conjunction with Espressif's official flashing accessories can be used for automated deployment and mass production flashing.

- Flashing baseboard (one module flashed at a time)
- Fixture (4 modules flashed at once)

Flashing process using Espressif' s official fixture

- 1. Relay power supply
 - The adapter is connected as shown in the figure to power the relay; the relay serves as a delay to ensure that the module is powered on after the fixture handle is pressed down stably.



2. The fixture baseboard serial port is connected to the computer through the HUB Use a USB cable to connect the 4 mini USBs on the two baseboards under the fixture box to the HUB. The HUB needs to be independently powered to ensure that each serial port provides sufficient power to the module; then connect the HUB to the computer. (The fixture can be connected to the computer for single serial port download)



3. Download

Select the firmware to be downloaded, set the serial port; place the modules in the four positions on the fixture, press down the handle, and click START to start downloading. For detailed steps, refer to: *Using the Flash Download Tool Flashing Process*

FactoryMult	Download														
				^	DownloadP	anel 1				DownloadP	anel 5				
			@		IDLE		^	PASS:	0 clear	IDLE		^	PASS:	0	clear
			🕫		等待			FAIL:	0	等待			FAIL:	0	clear
			@				×					×			
			@		START	STOP	ERASE	COM:		START	STOP	ERASE	COM:		~
			Ø					BAUD:	115200 ~				BAUD:	115200	~
			Ø		Deveload	anal 2				Deveload	anal 6				
			@		Download		A	PASS:	0	Download		A	PASS:	0	1
			@	~	IDLE 体化			FAIL:	0 clear	IDLE 体 在			FAIL:	0	clear
SPIFlashConfi	9				守付		\sim			守付		\sim			
SPI SPEED	SPI MODE	DoNotChgBin	DetectedInfo		START	STOP	EDACE	COM:	~	START	STOP	EDACE	COM:		~
④ 40MHz	QIO	2 LockSettings			START	3104	LIOASE	BAUD:	115200 ~	START	3104	LICASE	BAUD:	115200	~
O 26.7MHz		Cocksettings													
O 20MHz	OID ID	CombineBin			DownloadP	anel 3				DownloadP	anel 7				1
0 80MHz	ODUT	Default			IDLE		^	PASS:	0 clear	IDLE		^	PASS:	0	clear
	O FASTRD				等待			FAIL:	0	等待			FAIL:	0	
								COM		_			COM		
					START	STOP	ERASE	BAUD:	115200	START	STOP	ERASE	BAUD:	115000	
			~						115200					115200	
					DownloadP	anel 4				DownloadP	anel 8				
					IDI E		^	PASS:	0	IDLE		^	PASS:	0	alara
CT LOT					等待			FAIL:	0 Clear	等待			FAIL:	0	clear
ALL	A	LL ALL			4.14		~			9.19		~			
					START	STOP	ERASE	COM:		START	STOP	ERASE	COM:		~
								BAUD:	115200 ~				BAUD:	115200	\sim

Production Test Reference Materials

- Production Testing Guide
- Production Testing Tool

2.1.3 Environment Setup

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Espressif officially supports two types of development environments: ESP-IDF and Arduino IDE.

ESP-IDF

- ESP-IDF is an official development framework launched by Espressif, supporting Windows, Linux, and macOS operating systems.
- Development is conducted via command-line interface, enabling tasks such as configuration, compilation, flashing, and serial monitoring.
- For developers who prefer graphical interfaces, Espressif also provides **plugins for VSCode and Eclipse**, allowing development within the **IDE**'s **graphical environment**.

Arduino IDE

- The Arduino IDE is a versatile integrated development environment. It provides a specialized core for ESP series chips, encapsulating **part** of the ESP-IDF functions.
- It allows developers to utilize Arduino's simpler C++ programming language to develop ESP series chips.
- Developers do not need to have an in-depth understanding of the IDF details, making embedded programming more accessible, especially for beginners.

Selecting the Appropriate Development Environment For beginners and developers seeking simple and fast development:

• Arduino IDE is an excellent choice. Its user-friendly interface and basic functionalities allow for easy onboarding.

For developers with some programming experience:

• Visual Studio Code with the ESP-IDF plugin is a good choice. This combination provides a more friendly interface and automated configuration, making complex project development easier.

For advanced developers or professional developers familiar with Eclipse IDE:

- The ESP-IDF plugin for Eclipse will provide more flexibility and powerful features.
- Espressif-IDE provides a convenient ESP development environment.

For developers familiar with using the command line and seeking comprehensive features:

• ESP-IDF command line is a suitable choice. It is the official framework provided by Espressif, offering the most complete set of features to meet the needs of developers for complex embedded solutions.

ESP-IDF and Espressif Chips

下表总结了乐鑫芯片在 ESP-IDF 各版本中的支持状态,其中 😕 代表已支持, 💯 代表目前处于预览支持状态。预览支持状态通常有时间限 制,而且仅适用于测试版芯片。请确保使用与芯片相匹配的 ESP-IDF 版本。

芯片	v4.2	v4.3	v4.4	v5.0	v5.1	
ESP32	支持	支持	支持	支持	支持	
ESP32-S2	支持	支持	支持	支持	支持	
ESP32-C3		支持	支持	支持	支持	
ESP32-S3			支持	支持	支持	芯片发布公告
ESP32-C2				支持	支持	芯片发布公告
ESP32-C6					支持	芯片发布公告
ESP32-H2					支持	芯片发布公告

Click the title link to view the latest IDF version support information.

Official Development Environment Comparison

Arduino IDE

Suitable for:

• Beginners, entry-level developers, and users seeking simple and fast development

Advantages:

- Easy configuration and setup process
- Provides an intuitive user interface and simplified workflow, making it easy to write and upload code, which is convenient for rapid prototyping.
- Provides a wide range of libraries and ready-made codes, which are very convenient for rapid prototype validation and simple projects.
- Arduino IDE is easy to operates on multiple operating systems, including Windows, Mac, and Linux.
- Arduino IDE has extensive of documentation and tutorials for reference, making it easier to solve problems when encountered.

Disadvantages:

- Limited advanced features, cannot modify the underlying code. In complex ESP-IDF projects, using the original ESP-IDF development toolchain and command line tools can be more flexible.
- Limited debugging features. In complex ESP-IDF projects, other debugging tools are needed.

How to install:

• [Document] Install ESP32 Arduino Software Development Environment on Windows

ESP-IDF Command Line

Suitable for:

• Advanced developers. Developers with rich embedded system development experience, who have a certain understanding of embedded system concepts, low-level programming, and hardware-related knowledge. Developers who prefer or are interested in using command-line development environments.

Advantages:

- Provides a command-line based development environment using CMD or PowerShell, offering higher stability and flexibility.
- Developers can use scripts and command line tools in the command line development environment to batch build, test, and deploy applications, thereby improving development efficiency.
- The command line development environment is usually more lightweight and consumes less system resources. This makes it more suitable for development in resource-constrained embedded systems.
- Easy to install, with an official Windows one-click installation tool to quickly set up the environment.

Disadvantages:

- The command line development environment requires developers to have some knowledge of command-line and script programming. This presents a higher learning curve for beginners.
- Lack of graphical interface, which may not be as user-friendly and intuitive as a graphical interface. Not convenient for direct development.
- Need to edit code in other IDEs.

How to install:

• Windows One-click Installation Tool

- [Document] Windows Installation Tutorial
- [Document] Linux and macOS Platform Tutorial
- [Video Tutorial] Quickly Set Up ESP-IDF Development Environment on Windows Using One-Click Installation Tool

Visual Studio Code Extension

Target Audience:

• Developers with some embedded development experience who are already using or prefer to use VSCode as their main development environment.

Advantages:

- Integrates the ESP-IDF with VSCode, providing developers with the convenience of developing, building, and debugging ESP-IDF projects within the same interface.
- Offers simplified configuration features. Developers can easily configure project parameters, compilation options, and debugging settings, etc., through the plugin's configuration interface. This makes configuring the ESP-IDF environment and projects more straightforward and intuitive.
- VSCode is a relatively lightweight editor with fast startup speed and relatively low system resource consumption.

Disadvantages:

- Developers who are not familiar with VSCode or plugin development may need extra learning and adaptation.
- There may be limitations due to updates to the plugin itself and compatibility with new versions of ESP-IDF. Developers may need to monitor plugin updates and related documentation.
- Many functionalities depend on plugins.

How to Install:

- [English Document] Visual Studio Code Extension Installation Tutorial
- [Video Tutorial] Quickly Set Up ESP-IDF Development Environment Using VS Code (Windows, Linux, MacOS)

Espressif-IDE or Eclipse Plugin

Note: Espressif-IDE comes with IDF Eclipse Plugin, basic Eclipse CDT plugins, and other third-party plugins of the Eclipse platform to support the building of ESP-IDF applications.

Target Audience:

• Advanced and professional developers. Developers with embedded system development experience who are already using or prefer to use Eclipse.

Advantages:

- Integrates the ESP-IDF framework with Eclipse, providing developers with the convenience of developing, building, and debugging ESP-IDF projects under the same interface.
- Offers superior support for complex large-scale projects.
- Eclipse IDE has a vast ecosystem of plugins, allowing developers to install and use other useful plugins according to project requirements, thereby enhancing development efficiency.
- It possesses powerful graphical debugging capabilities.

Cons:

• Compared to the aforementioned lightweight development environments or pure command-line environments, Eclipse IDE itself may consume more system resources, especially on lower-performance computers, it may feel slower.

- Eclipse is relatively complex in terms of environment configuration, and it is not very friendly to beginners.
- Eclipse has a fairly large user community, but it is smaller compared to other environments.

How to install:

- [Document] Espressif-IDE
- [Document] Eclipse Plugin Tutorial
- [English Document] Eclipse Plugin Windows Installation Tutorial
- [Video Tutorial] Eclipse Plugin Tutorial (English)

Note: For issues with environment setup, you can refer to csdn blog

The choice of the appropriate development environment depends on your level of experience, project complexity, and personal preference. Beginners can start with Arduino IDE, then gradually transition to the plugin-equipped VSCode environment. Experienced developers can directly use ESP-IDF or use the ESP-IDF plugin in Eclipse for more complex project development.

Other Development Environments

1. CLion

CLion is an IDE developed by JetBrains, specifically designed for developers using C and C++ programming languages. It offers intelligent code completion, syntax highlighting, code navigation, powerful refactoring capabilities, and debugging tools to enhance developer efficiency. CLion also integrates version control systems, such as Git, to facilitate team collaboration.

2. CircuitPython

CircuitPython is an open-source version of the Python language designed for beginners, specifically for small, cost-effective computers known as microcontrollers. Microcontrollers are at the heart of many electronic devices, including various development boards used for building hobby projects and prototypes. CircuitPython simplifies the process of experimenting and learning programming on these low-cost microcontroller development boards, making it easier for beginners to get started.

3. MicroPython

MicroPython is a highly simplified interpreter based on Python, designed specifically for embedded systems. It allows developers to use the familiar Python language for embedded development, making programming simpler and more efficient. MicroPython can run on a variety of hardware platforms, including ESP series chips.

4. PlatformIO IDE

PlatformIO is a cross-platform open-source ecosystem that supports multiple hardware platforms and development boards, and provides a unified development environment, allowing developers to use different hardware and development boards for embedded development. PlatformIO integrates multiple development tools, including compilers, debuggers, and upload tools.

5. Toit

Toit is a modern memory-safe programming language. It features advanced editor integration, including functions such as syntax highlighting, jump-to-definition, and auto-completion.

6. UIFlow

UIFlow is an easy-to-use graphical programming IDE that anyone can get the hang of. It supports both wireless and wired program pushing, allowing programs to run with just a click, eliminating the need for repeated compilation. It supports over 100 M5 hardware peripherals

and sensors, and allows for one-click addition of extensions, effectively aiding in product prototype construction and accelerating the development process to final productization.

7. Wokwi

Wokwi is an online ESP32 simulator that allows developers to simulate and evaluate some ESP32 software projects without ESP32 hardware, providing a faster solution for the prototype design of IoT projects for the ESP32 microcontroller.

Attention: The platforms listed in this section are not directly developed and maintained by Espressif. If developers encounter difficulties during use, please seek support in the corresponding community and forum.

Third-Party Reference Materials

Arduino Windows

• [Video] Arduino ESP32 Environment Setup Guide

Arduino Ubuntu

• [Video] ESP32 Development Environment Setup (Arduino)

VSCode

- [Video] ESP32-IDF Environment Setup with VSCode
- [Video] VSCode + ESP-IDF Development Environment Setup (Implementing Step-by-Step Debugging)

Eclipse

• [English Video] Install ESP32 + Eclipse IDE 2020 (English)

2.1.4 Getting Started with Project Development

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

When developing an ESP project, the typical process begins with understanding the relevant ESP examples and SDK based on the project's requirements. The next step is to select the most suitable example as a template for the target project, and then modify it to create your own project.

This article will summarize essential knowledge required for developing general embedded projects within the Espressif environment:

- General Knowledge
- Understanding ESP Series Chips and ESP-IDF Framework
- Summary of Project Development Knowledge Points
- Learning Methods

General Knowledge

Before developing an ESP project, some key general knowledge and skills are needed to ensure smooth development:

• Git:

Git is an open-source distributed version control system used to track code changes and collaborative development. Using Git in your project can help you manage your code effectively, control versions, and work collaboratively with team members. Learning how to use Git for code submission, branch management, merging, etc., is a basic requirement for developing ESP projects. Here, developers are recommended to master some common Git commands

• FreeRTOS:

FreeRTOS is an open-source real-time operating system widely used in embedded systems and microcontroller projects. ESP series chips all support FreeRTOS, and often use it to implement multitasking and real-time scheduling. Understanding the concepts of FreeRTOS task scheduling, message queues, semaphores, etc., is essential knowledge for developing ESP projects.

- Developers can learn from the official FreeRTOS getting started guide.
- Learn how to use FreeRTOS in the ESP-IDF environment in the FreeRTOS section of the ESP-IDF Programming Guide.
- Linux (optional):

ESP project development often takes place on the Linux operating system, as Linux provides a wealth of tools and command-line environments suitable for embedded development. Developers should understand the basic commands and usage of Linux, such as file operations, directory management, process management, etc., in order to debug and configure during development. Developers are recommended to master some common Linux file and directory management commands

```
- ls: Lists files and folders in the current directory.
- cd <directory>: Switches to the specified directory.
```

- pwd: Displays the path of the current working directory.
- cp <source> <destination>: Copies files or directories.
- rm <file name>: Deletes a file.
- mkdir <directory>: Creates a new directory.
- rmdir <directory>: Deletes an empty directory.
- cat <file_name>: Displays the contents of a file.

Understanding ESP Series Chips and ESP-IDF Framework

- For the features and functions of ESP chips, refer to: Board selection
- Introduction and Management of ESP-IDF Versions
- Composition and Architecture of the ESP-IDF Framework
- Setting and Configuring the Development Environment
- Using ESP-IDF Tools
- Using ESP-IDF API and Guide to ESP-IDF API Usage
- Security Features of ESP Chips

Recommended Websites:

- ESP-IDF Quick Start
- ESP-IDF Programming Guide

Summary of Project Development Knowledge Points

C Language

- 1. Data Types
 - Basic Data Types
 - Integer, Character, Floating Point
 - Standard Library Data Types
 - Boolean (bool), String (string)

- Pointer Types
- Composite Data Types
 - Array, Structure (struct), Enumeration (enum)
- Custom Data Types
- Data Type Qualifiers
 - const, volatile
- 2. Functions, Pointers, and Memory Management
- 3. Compilation, Linking, and Execution Process of Engineering Projects
- 4. Algorithms
 - Data Structures
 - Linked List
 - Stack and Queue
 - Tree and Graph
 - Sorting Algorithms
 - Bubble Sort
 - Selection Sort
 - Insertion Sort
 - Quick Sort
 - Merge Sort
 - Search Algorithms
 - Linear Search
 - Binary Search
 - Hashing
 - Recursive Algorithms
 - Divide and Conquer
 - Backtracking
 - Dynamic Programming
 - Encoding and Decoding Algorithms
 - Handling Data Compression, Encryption, Encoding, and Decoding
 - Signal Processing Algorithms
 - Processing of audio and images
 - Algorithm optimization
 - Time complexity
 - Space complexity

Communication Protocols

- 1. Master various communication protocols, such as UART, SPI, I2C, CAN, etc., for communication with external devices
- 2. Learn network protocol stacks, such as **TCP/UDP**, as well as higher-layer application protocols, such as **HTTP**, **MQTT**, etc.

Project Building and Management

- 1. Build System
- 2. Partition Table
- 3. Component Management and Usage

Application Programming

- 1. GPIO (General Purpose Input/Output), General Input/Output
 - GPIO initialization, mode, reading, writing
 - GPIO configuration options
 - Determine the function of each pin according to the datasheet and pin diagram
 - GPIO interrupts
 - Strapping pins
- 2. JTAG (Joint Test Action Group) Debugging

- 3. Memory Management
 - Dynamic memory allocation and release, such as malloc and free
 - Memory layout and stack management
- 4. Interrupt Handling
 - Understand and handle hardware interrupts
 - Implement Interrupt Service Routines (ISR) to respond to external events
- 5. Clocks and Timers
 - Use timers and clock sources to implement time control and timed tasks
 - Handle delay and timing operations
- 6. Exception Handling
 - Handle hardware and software exceptions
- 7. Low Power Mode Design
 - Implement power optimization strategies to extend battery life or reduce energy consumption
- 8. Processes
 - Creation and termination of processes
 - Management and scheduling of processes
 - Resource allocation and usage of processes
 - Methods of inter-process communication
 - Pipes, Named Pipe (FIFO), Message Queue, Signal, Shared Memory, Semaphore, Socket
- 9. Threads
 - Creation and destruction of threads
 - Scheduling and synchronization of threads
 - · Concurrency control in multithreaded programming
 - Methods of inter-thread communication

Mutex, Condition Variable, Semaphore, Barrier, Message Queue, Shared Memory, Spin Lock

- 10. Driver Development
 - Drivers can be written for various hardware devices, including but not limited to sensor, actuator, storage devices (such as flash memory and SD cards), communication interfaces (such as UART, SPI, I2C), display, network interface card (NIC), etc.

Learning Methods

- Online resources and documents:
 - Utilize the ESP-IDF official documentation, tutorials, and example codes to gain a deep understanding of the framework and API usage.
 - Find ESP-IDF based solutions, application examples, components, and drivers in the ESP IoT Solution library. Most documents provide both Chinese and English versions.
- Online courses and video tutorials:
 - Learn relevant knowledge and practical skills by participating in online courses and video tutorials on ESP chip and ESP-IDF development.
- Experiments and projects:
 - Deepen your understanding of hardware and software by using ESP development boards for experiments and project development. Beginners can learn from other completed projects or solutions released by Espressif.
- Community and forums:
 - Join the ESP32 official forum, CSDN forum, or other developer communities. Exchange experiences with other developers, seek help, and share projects and solutions.
 - Submit bugs or feature requests through the Issues section on GitHub. Please check the existing Issues before submitting new ones.
- Reference books:
 - Read books related to embedded systems, C language, and ESP chip development to expand your knowledge breadth and depth. Reference: ESP32 book list.

Continuous practice and project development are the most important parts of the learning process. Developers can gradually master the skills of embedded project development with ESP chips through continuous practice and practical application.

2.2 Advanced Development

This section is suitable for developers who wish to gain a deeper understanding and mastery of more advanced programming techniques. We will introduce some more complex development techniques, including component management, code debugging, etc., and this section will provide in-depth guidance.

2.2.1 Component Management and Usage

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Imagine you are developing a complex application that involves multiple functional modules, such as network communication, sensor reading, and screen driving. If you write all functions in one large file, the code will become lengthy and difficult to maintain. At this point, components play a role in modularization.

Each component focuses on a specific functional area, so each component can be developed, tested, and maintained separately. By breaking down functions into components, you can reuse these functions in multiple projects without having to rewrite code. This not only saves development time but also improves development efficiency. The structure of the project will also become clearer.

- Finding Components
- Adding Components with Component Manager
- Using Local Components
- Pulling Components from Git Repository
- Updating Components
- Deleting Components
- Developing Components (Advanced)
- Releasing Component

IDF Components

ESP-IDF components are reusable code packages that are compiled into static libraries during build and can be linked by other components or applications. This allows them to be used in multiple projects.

A complete component generally includes:

- Source code
- Header files
- CMakeLists.txt
 - Defines source code and header files
 - Defines dependencies
 - Registers components
 - Configures optional features
 - CMake build description file, used to describe the build configuration and dependencies of the component, instructing the compiler how to compile, link, and build the component.
- idf_component.yml
 - Component manager description file, which lists the other components to be referenced and their version information. In this way, when building a project, the ESP-IDF component manager will automatically download and integrate the required components according to these descriptions to ensure that the project' s dependencies are met.

Component Dependencies

When compiling each component, the ESP-IDF system will recursively evaluate its dependencies. This means that each component needs to declare the components it depends on, i.e., "requires".

A dependency refers to other software libraries, modules, or components that a software project depends on. These dependencies are necessary for building and running the project. Software projects often use existing code or libraries to implement specific functions or provide certain services, rather than writing all the code from scratch. These external codes or libraries are the dependencies of the project.

The role of dependencies is to promote code reuse, reduce development costs, speed up development, and improve software quality. Using existing dependencies can avoid reinventing the wheel.

Common dependencies include:

- · Third-party libraries
- Frameworks and components
- Runtime environment

Finding Components

- 1. IDF includes some basic functional components, which can be found in the components directory
- 2. idf-extra-components includes some IDF supplementary components
- 3. esp-iot-solution provides some peripheral driver components
- 4. Components maintained by Espressif and those uploaded by third parties can be found in the esp-registry.
- 5. Searching for third-party component libraries

Component Manager

The Component Manager refers to the tool or system used for managing components. In IDF, the Component Manager provides a set of commands for adding, removing, downloading, and managing components, so as to reuse existing functional modules in the project.

Adding Components with Component Manager

- 1. Find the components needed for the project in the esp-registry.
- 2. Use the terminal to enter the root folder of the project directory and execute idf.py add-dependency
 "[namespace]/<component name>[version number]".
 - This command will automatically add an entry to main/idf_component.yml, and if the file does not exist, it will be created automatically.
 - Namespace: Optional, default from espressif (case insensitive)
 - Component name: Required, the name of the component, for example usb_stream (case insensitive)
 - Version number: Optional, default is * representing any version. When adding a component for the first time, the latest version will be downloaded automatically.
- 3. Execute idf.py build

Note:

- No need to make additional modifications to the project CMakeLists.txt, you can directly include the public header files of the component in the project source code, compile and link the static library of the component.
- Users cannot directly modify the components in managed_components, because the Component Manager will automatically detect the component Hash value and restore changes.
- In addition to using the command idf.py add-dependency, users can also directly modify the idf_component.yml file and manually modify the component entries.

Using Local Components

In the Component Manager description file idf_component.yml, add the local component address as shown in the example below.

```
dependencies:
    idf: ">=4.4.1"
    cmake_utilities: "0.*"
    iot_usbh_modem:
        version: "0.1.*"
        override_path: "../../../components/usb/iot_usbh_modem"
```

Pulling Components from Git Repository

In the Component Manager description file idf_component.yml, add the component's address on Github as shown in the example below.

```
dependencies:
    esp-gsl:
    git: https://github.com/leeebo/esp-gsl.git
    version: "*"
    button:
        git: https://github.com/espressif/esp-iot-solution.git
        path: components/button
        version: "*"
```

Updating Components

- 1. Modify the component version in the component manager description file idf_component.yml to the target version you want to update to
- 2. Delete managed_components and dependencies.lock in the project file
- 3. Execute idf.py build or manually execute idf.py reconfigure

Deleting Components

- 1. Delete the component entry in idf_component.yml
- 2. Delete build, managed_components and dependencies.lock
- 3. Execute idf.py build

Developing Components (Advanced)

- 1. Create a component directory, consistent with ESP-IDF component requirements, and write the corresponding CMakeLists.txt file
- 2. Add idf_component.yml file, add component information, IDF version requirements, specify dependent other components
- 3. Add license.txt file, add license information of the component. The license information is used to explicitly inform other developers or users of the terms and conditions that need to be complied with when using this component. This can ensure that the use and distribution of the component is legal, and clarify the copyright and authorization status of the component author for his work. The license information is generally contained in a file named license.txt or LICENSE, which includes the following content:
 - License type: Explicitly specify the license type of the component, such as MIT license, Apache license, GPL license, etc. Different types of licenses have different terms and conditions, and developers and users need to comply with the corresponding license regulations.
 - Copyright information: Indicate the copyright ownership of the component, that is, the author or copyright holder of the component. Copyright information usually includes the author's name, organization or company name, and copyright year.

- Disclaimer: May contain a disclaimer, stating that the author or copyright holder of the component does not assume any liability for any loss or liability that may be caused by the use of the component.
- Permissions and restrictions: Explicitly specify under what conditions the component can be used and distributed, as well as prohibited behaviors or restrictions.
- Open source license: If the component is open source software, the license information should contain the corresponding open source license terms, such as the rules for sharing, modifying, and distributing open source code.
- 4. Add README.md file, add a brief introduction of the component, simple usage instructions
- 5. Add CHANGELOG.md file, add the version update record of the component
 - A document used to record the version update record of the component. It usually contains the changes, improvements, fixes, and new features of each version of the component. Developers and users can understand the version update history of the component by referring to the CHANGELOG.md file, so as to better understand the development and use of the component.
- 6. Add test_apps directory, add test cases of the component
 - The component's self-check tool, which can ensure the quality and stability of the component, is very important for the development and maintenance of the component
 - It contains applications that test different aspects or functions of the component
- 7. Add examples directory
 - Provide example code for the component to demonstrate the basic usage and functions of the component
 - Or specify the examples path in idf_component.yml, add example code of the component

Releasing Component

Release on ESP-Registry ESP-Registry is a central repository provided by Espressif, offering developers a convenient way to discover and download components for their engineering projects.

- 1. Register an account on ESP-Registry You can complete the registration on ESP-Registry by authorizing with your Github account
- 2. After registering, click on the username in the top right corner and select Tokens
- 3. On this page, click Create to create a Token
- 4. Register the newly created Token to the environment variable IDF_COMPONENT_API_TOKEN
- 5. Use the idf.py upload-component command to upload components to the ESP component registry - idf.py upload-component --namespace [YOUR_NAMESPACE] --name test_cmp -YOUR_NAMESPACE: ESP-Registry username, which can be seen in the profile - test_cmp: The name of the component to be uploaded
- 6. After the upload is complete, the component can be viewed on the ESP-Registry.

Published on Github Repository If you only want to publish to the Github Repository, you can follow the component development specifications and directly publish the component to your own Github Repository. Other developers can pull this component using the previous *Pulling Components from Git Repository* method.

Related Links

- Component Manager Github
- ESP-Registry Website
- ESP-Registry Documentation
- upload-components-ci-action

2.2.2 Advanced Code Debugging

This section introduces common software exception crashes on ESP chips, as well as some commonly used advanced code debugging methods when encountering similar problems.

Overview: Common Panic & Exception Introduction

Before debugging the code, it is necessary to understand the common Panic & Exception, which include the following situations:

- Watchdog Interrupt
- Brownout Interrupt
- Assert / Abort
- Stack Overflow
- Cache Access Error
- Invalid Memory / Instruction Address

Watchdog Interrupt This part often involves two situations: interrupt watchdog and task watchdog. For detailed documentation, please refer to Watchdog. The following are brief descriptions of these two watchdogs:

Interrupt Watchdog The purpose of the interrupt watchdog timer is to ensure that the interrupt service routine (ISR) is not blocked for a long time (i.e., IWDT timeout). Blocking the timely execution of the ISR will increase the ISR delay and also prevent task switching (because task switching is executed from the ISR). Matters that prevent the ISR from running include:

- Disabling interrupts
- Critical section (also disables interrupts)
- Other ISRs of the same or higher priority will prevent ISRs of the same or lower priority from completing

When the IWDT times out, the default operation is to call the panic handler and display the error cause (Interrupt wdt timeout on CPU0 or Interrupt wdt timeout on CPU1, depending on the situation).

At this time, the above three points need to be investigated, especially whether too much code is placed in the ISR, causing the ISR time to be too long and triggering the interrupt watchdog.

For more detailed information and analysis methods about the interrupt watchdog, please refer to the interrupt watchdog section in *Watchdog*.

Task Watchdog The task watchdog timer is used to monitor whether the tasks in FreeRTOS are scheduled within a specified time. If the default lowest priority IDLE task does not feed the dog (i.e., reset the watchdog timer) within the specified time, it will trigger the watchdog interrupt. This usually indicates that a high-priority task is always occupying the CPU due to reasons such as calling an API in an infinite loop without delay.

The debugging method for this problem is briefly illustrated as follows, deliberately writing an infinite loop without delay in *app_main* to print the log:

After the ESP is powered on and runs for CONFIG_ESP_TASK_WDT_TIMEOUT_S seconds, you can see the following log:

```
E (10303) task_wdt: Task watchdog got triggered. The following tasks/users did not

→reset the watchdog in time:

E (10303) task_wdt: - IDLE (CPU 0)

E (10303) task_wdt: Tasks currently running:

E (10303) task_wdt: CPU 0: main

E (10303) task_wdt: CPU 1: IDLE

E (10303) task_wdt: Print CPU 0 (current core) backtrace
```

The above log can be briefly analyzed as:

- *E* (10303) *task_wdt: IDLE* (*CPU* 0) -> IDLE0 on CPU0 did not feed the dog in time
- *E* (10303) task_wdt: CPU 0: main -> The task currently running on CPU0 is the main task (the main task corresponds to the app_main function)
- *E*(10303) task_wdt: CPU 1: IDLE -> The task currently running on CPU1 is the IDLE1 task

At this time, it is indicated that the IDLE0 task triggered the watchdog reset because it did not feed the dog in time within CONFIG_ESP_TASK_WDT_TIMEOUT_S seconds, and then it can be inferred that the main task should always occupy the CPU to cause the task watchdog reset through the task running on CPU0. This task should be the focus of investigation, and appropriate delays should be added to give other low-priority tasks the opportunity to be scheduled.

For more detailed information and analysis methods about the task watchdog, please refer to the task watchdog section in *Watchdog*.

Brownout Interrupt This part is a brownout interrupt. The ESP chip integrates a brownout detection circuit internally and is enabled by default. When the brownout detector is triggered, the following information will be printed:

Brownout detector was triggered

The chip will reset after the print information ends. At this time, you should check whether the hardware power supply voltage meets the set threshold. For more information, please refer to the Brownout document.

Note: In the scenario of battery power supply, such as 2xAA battery power supply, the voltage is 3.1 V. At this time, the voltage will drop for a short time due to the large instantaneous current in scenarios such as Wi-Fi connection, triggering the brownout detection and causing the chip to restart.

It is recommended to use a voltage regulator chip with a larger peak current. Or replace with a battery that can provide a large current, or try to increase the capacitance of the power supply.

Assert / Abort This part is triggered by assertion or abortion. Assertions are commonly used to check whether assumptions in the program are true. If an assertion fails (i.e., the assumption is not established), it will trigger an interrupt. After the interrupt, the program usually aborts (abort) and records error information in the log to help developers debug.

Therefore, you can check which API triggered the assertion or abortion in the log at this time, and debug the code from this API, for example:

```
// Function to check if input is equal to 1
int check_input(int input) {
    if (input == 1) {
        return 1; // Return 1 if input is equal to 1
    }
    else {
        return 0; // Return 0 if input is not equal to 1
    }
}
void app_main(void)
{
    // Test case for input not equal to 1
    int input1 = 2;
    assert(check_input(input1) == 1); // Assert that the function returns 1 for-
    input 1
}
```

The corresponding exception log is:

assert failed: app_main hello_world_main.c:26 (check_input(input1) == 1)

It can be seen that the reason is that $assert(check_input(input1) == 1)$ triggered the assertion because the condition was not satisfied. At this time, you can focus on investigating the code corresponding to this assertion.

Note: Since assertions can be disabled in the CONFIG_COMPILER_OPTIMIZATION_ASSERTION_LEVEL option in ESP-IDF menuconfig, it is not recommended to do calculations or function calls in assertions.

Stack Overflow This part is stack overflow. When the task stack actually used exceeds the pre-allocated task stack, this error will occur. The error example code is as follows:

```
static void test_task (void *pvParameters)
{
    uint32_t large_array[4096] = {0};
    while (1) {
        // This task obstruct a setting tx_done_sem semaphore in the UART_
        -interrupt.
        // It leads to waiting the ticks_to_wait time in uart_wait_tx_done()_
        -function.
        vTaskDelay(200 / portTICK_PERIOD_MS);
    }
    vTaskDelete(NULL);
}
void app_main(void)
{
    xTaskCreate(test_task, "test_task", 1024, NULL, 5, NULL);
}
```

The corresponding error log is as follows:

ERROR A stack overflow in task test_task has been detected.

From the log, you can see that a stack overflow occurred in the test_task task. Checking the code, you can see that only 1024 bytes of task stack were allocated when *xTaskCreate*, but a 4096-byte array is used in the task, which caused the stack overflow. At this time, you need to reduce the array size in the task function, or increase the task stack allocated by *xTaskCreate*.

Cache Access Error For this part, you can first refer to the Cache disabled but cached memory region accessed document. More detailed explanations of this error will be given in *Locating Problems Using Guru Meditation Error Printing*.

Invalid Memory / Instruction Address When the program tries to access a non-existent memory address or execute an invalid instruction, this type of exception will be triggered. This is usually caused by pointer errors or memory corruption. When this type of error occurs, you can refer to *Locating Problems Using Backtrace & Coredump* and *Locating Memory Issues (Memory Stompings, Memory Leaks, etc.)* chapters for further code debugging.

Overview: Panic Handler and Common Code Debugging Methods Introduction

Panic Handler is described in detail in the ESP-IDF Programming Guide, and will not be repeated here.

The common code debugging methods are as follows.

Log Printing / App Trace ESP-IDF provides a convenient and easy-to-use log printing library ESP Logging and App trace.

Backtrace / Core Dump For detailed guidance on this part, please refer to *Locating Problems Using Backtrace & Coredump*.

GDB Stub through UART Please refer to GDB Stub, which can be briefly summarized as follows.

GDB Stub Postmortem

- 1. Enable *CONFIG_ESP_SYSTEM_PANIC_GDBSTUB* so that when an error occurs, esp_monitor will automatically enter GDB through UART
- 2. Use ELF + Core dump records, enter GDB through *idf.py coredump-debug*

Common GDB commands:

- 1. *backtrace* to trace the call stack (including incoming parameters)
- 2. *list* to print the code location
- 3. *info threads* to print all Task information
- 4. info locals to print the local variables of the current Task
- 5. *print <var>* to print the local/global variables of the current Task
- 6. *thread <id>* to switch Task

You can also refer to GDB Cheat Sheet.

GDB Stub Debugging

1. Enable *CONFIG_ESP_SYSTEM_GDBSTUB_RUNTIME*, you can automatically enter GDB through UART esp_monitor at runtime (*Ctrl*`+`C)

Common GDB commands:

- 1. *x/lxw <addr>* to print the corresponding 32-bit memory/register address
- 2. watch <var> to monitor variables, when the variable value changes, GDB will automatically stop
- 3. *break <where>* to set breakpoints, which can be function names, line numbers, addresses
- 4. continue to continue running
- 5. *step* to run step by step, enter the function
- 6. *next* to run step by step, do not enter the function
- 7. *print <var>* to print variable values
- 8. *set <var> = <value>* to modify variable values

You can also refer to GDB Cheat Sheet.

OpenOCD & GDB Please refer to JTAG Debugging.

GPIO tracing GPIO tracing often flips IO to mark events, such as implementing GPIO flipping when a certain event occurs in the code, so that you can determine whether the event is triggered by querying the flipping status of GPIO.

SystemView through UART/JTAG For detailed guidance on this part, please refer to *Using SystemView for System Analysis and Optimization*.

Locating Problems Using Guru Meditation Error Printing

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

You can first refer to the Guru Meditation Errors in the ESP-IDF Programming Guide to understand common errors and their causes. This document provides further explanations of common errors and presents feasible analytical approaches.

Note: This analysis focuses on errors occurring in the RISC-V architecture, specifically the ESP32-C3. While errors in the Xtensa architecture (e.g., ESP32) may differ in naming, their underlying causes remain similar. Architecture-specific errors will be analyzed separately.

Guru Meditation errors are categorized into two types: ARCH exceptions and SOC exceptions.

Common ARCH exceptions:

- Load access fault
- Store Access Fault
- Instruction Access Fault
- IllegalInstruction

Common SOC exceptions:

- Interrupt wdt timeout on CPU0/CPU1
- Cache disable but cached memory region accessed

Often, identifying the root cause of Guru Meditation errors can be challenging, as these crashes often represent symptoms of underlying issues. The actual problems may be related to FreeRTOS, memory, flash, interrupt, etc., requiring specific analysis.

Load access fault (LoadProhibited)

Test Case This exception occurs when a load instruction (such as lw) attempts to access invalid memory. The example code demonstrates a scenario that triggers this crash:

```
uint8_t *buff = NULL;
load_fault = buff[1];
```

Exception Phenomenon

• RISC-V

```
Guru Meditation Error: Core 0 panic'ed (Load access fault). Exception was_
\hookrightarrowunhandled.
Core 0 register dump:
Stack dump detected
MEPC : 0x4200696a RA : 0x4200696a SP
                                                : 0x3fc8f7c0 GP
                                                                     : __
⇔0x3fc8b400
   : 0x3fc87d74 T0 : 0x4005890e T1
                                                : 0x3fc8f41c T2
TΡ
                                                                     ↔0x00000000
S0/FP : 0x3c023000 S1 : 0x0000000 A0
                                                : 0x000000f A1
                                                                     ⇔0x3fc8f3f8
A2 : 0x0000000 A3
                            : 0x0000001 A4
                                                : 0x3fc8c000 A5
                                                                     . ...
→0x0000000
```

(continues on next page)

						(continued fro	m previous page)	
A6	: 0x60023000	Α7	: 0x000000a	S2	: 0x0000000	S3	:	
		a F	0 0000000	9.6	0 0000000	07		
54	: UXUUUUUUU	55	: UXUUUUUUU	56	: 0x00000000	57	:	
→0X0000	00000							
S8	: 0x0000000	S9	: 0x0000000	S10	: 0x0000000	S11	:	
↔0x0000	0000							
Т3	: 0x0000000	Τ4	: 0x0000000	Т5	: 0x00000000	Т6	:_	
→0x0000000								
MSTATUS	: 0x00001881	MTVEC	: 0x40380001	MCAUSE	: 0x0000005	MTVAL	:_	
⇔0x000(↔0x00000001							

• XTENSA

Guru Meditation Error: Co	ore 0 panic'ed (L	oadProhibit	ted). Exception	on was unhandled.					
Core 0 register dump:									
PC : 0x42007c2f PS	: 0x00060a30	A0	: 0x82007c42	A1 :					
⊶0x3fc97d70									
A2 : 0x42011934 A3	: 0x3c023f48	A4	: 0x3c023fc8	A5 :					
⇔0x3fc97d90									
A6 : 0x3fc97d70 A7	: 0x000000c	A8	: 0x00000000	A9 :					
⇔0x3fc97d20									
A10 : 0x000000f A11	: 0x3fc97fac	A12	: 0x3fc97d70	A13 :					
→0x000000c									
A14 : 0x3fc941e4 A15	5 : 0x0000000	SAR	: 0x0000004	EXCCAUSE:					
⇔0x000001c									
Backtrace: 0x42007c2c:0x3fc97d70 0x42007c3f:0x3fc97d90 0x42019247:0x3fc97db0_ →0x40379e71:0x3fc97de0									

Troubleshooting If further analyze the assembly code (the steps can be referred to *Appendix 1 - ESP Disassembly Corresponding Instructions*), you can find it corresponds to the following instructions:

lw a1, 1(zero)	// RSIC-V
18ui a11, a8, 1	// XTENSA

When this type of exception is triggered, MTVAL/EXCVADDR will automatically update to data related to the exception. From the RISC-V exception error, you can see MTVAL : 0x00000001.

Usually, MTVAL being NULL(0x0000000) represents taking a value from a NULL address, and close to NULL represents trying to access a member of an array or structure from a NULL address.

- For the RSIC-V architecture, the PC and RA registers contain the function pointer of the exception, and you can find the corresponding function from the elf.
- For the Xtensa architecture, a backtrace will be printed during an exception, and the corresponding function can also be found from the elf.

Practical Case The following figure is a typical ESP32-C3 Load access fault issue.

				AND DIRIGIAN NICTAD TREAMING CHINARD CHINARD
4038fae6:	080c	addi	a1,sp,16	46 I (16977) write data: 7f 03 00 12 3f 01 00 00 00 00 40 01 37 15 01 00
40381ae8:	31Ca4/b/	lui	a5,0X3TCa4	47 I (16983) write data: 3f 01 3d 40 00 40
4038Taec:	35878503	LW 4-2	a0,856(a5) # 3TCa4358 <xilmerqueue></xilmerqueue>	48
40381810:	adcrower	jau	ra,4038cdcc <xqueuekeceive></xqueuekeceive>	49 phhalHw_Pn5190_InstMngr_SendSnglTlv end
40381814:	C5/1	beqz	a0,4038FDC0 <prvprocessreceivedcomma< td=""><td>50 Now wait for the IRQ</td></prvprocessreceivedcomma<>	50 Now wait for the IRQ
05+0X12>	47-0		-5 46()	51 Guru Meditation Error: Core 0 panic'ed (Load access fault). Exception was unhandled.
40381810:	4/C2	LW beez	d5,10(SP)	152
40301010.	100/0305	byez	as,40381ade «profilocesskecetvedcomma	53 Core 0 register dump:
40295556	hfa0		4829fad6 corvProcessPeceivedCommands	54 MEPC : 0x4038fb00 RA : 0x4038faf4 SP : 0x3fcae860 GP : 0x3fc99a00
4038181C.	UTES		40381800 Cpr VFT OCESSRECE CVEUCOMMands	55 0x4038fb00: prvProcessReceivedCommands at D:/esp_king/esp-idf20230608/esp-idf/components/freer
4038fafe'	4462	٦w	sA 24(sp)	56
4038fb00:	4850	lw	a5 20(s0)	57 0x4038faf4: prvProcessReceivedCommands at D:/esp_king/esp-idf20230608/esp-idf/components/free
4038fb02	c789	heaz	a5 4038fb0c <prvprocessreceivedcomma< td=""><td></td></prvprocessreceivedcomma<>	
ds+0x3e>		bede		59 TP : 0x37688454 T0 : 0x000000000 T1 : 0x000000000 T2 : 0x000000000
4038fb04:	00440513	addi	a0.s0.4	61 32 · 0 x00000000 51 · 0 x00000000 A0 · 0 x00000000 A1 · 0 x00000000 A
4038fb08:	bf5fc0ef	ial	ra,4038c6fc <uxlistremove></uxlistremove>	101 AZ . UA4USACSTA AS . UASICACADA A4 . UASICACADA A5 . UAUUUUUUU4
4038fb0c:	0068	addi	a0.sp.12	63
4038fb0e:	ebbff0ef	jal	ra,4038f9c8 <prvsampletimenow></prvsampletimenow>	64.46 : 0x00000000 A7 : 0x00000000 S2 : 0x00000000 S3 : 0x00000000
4038fb12:	47c2	lw	a5,16(sp)	65.54 : 0x00000000 55 : 0x00000000 56 : 0x00000000 57 : 0x000000000
4038fb14:	4715		a4,5	66 S8 : 0x00000000 S9 : 0x00000000 S10 : 0x00000000 S11 : 0x00000000
4038fb16:	08e78863	beq	a5,a4,4038fba6 <prvprocessreceivedco< td=""><td>167 T3 : 0x00000000 T4 : 0x00000000 T5 : 0x00000000 T6 : 0x00000000</td></prvprocessreceivedco<>	167 T3 : 0x00000000 T4 : 0x00000000 T5 : 0x00000000 T6 : 0x00000000
mands+0xd8>				68 MSTATUS : 0x00001881 MTVEC : 0x40380001 MCAUSE : 0x00000005 MTVAL : 0x00000014
4038fb1a:	04f74963	blt	a4,a5,4038fb6c <prvprocessreceivedco< td=""><td>0 69 0x40380001: vector table at ??:?</td></prvprocessreceivedco<>	0 69 0x40380001: vector table at ??:?
mands+0x9e>				170
4038fb1e:	470d	li	a4,3	71 MHARTID : 0x00000000
4038fb20:	06e78d63	beq	a5,a4,4038fb9a <prvprocessreceivedco< td=""><td>172 Backtrace: 0x4038fb00:0x3fcae860 0x4038fbda:0x3fcae890 0x4038fcd8:0x3fcae8b0</td></prvprocessreceivedco<>	172 Backtrace: 0x4038fb00:0x3fcae860 0x4038fbda:0x3fcae890 0x4038fcd8:0x3fcae8b0
mands+0xcc>				73 0x4038fb00: prvProcessReceivedCommands at D:/esp_king/esp-idf20230608/esp-idf/components/freer
tranoct/8.tcoin mb	ion meal:		24524,1 5%	174
				4/5 0X4038TDda: prvlimerlask at D:/esp_king/esp-idt20230608/esp-idt/components/freertos/timers.c:
	JOKJ),			470 Olde Test a Teb Width 0 a
st_fromisr(&tcpip_m	ibox, msg);			Plain Text * Tab Width: 8 *
w(&tcpip_mbox, TCP	IP_MBOX_SIZ ×			
<pre>&tcpip_mbox, TCPIP_</pre>	MBOX_SIZE) != ERR	2		e rs.pxTimer; ¬ Zim Kalinowski, 2年前 • freertos: Sync safe changes from Amazon SMP branch
ents/lwip/lwip/src/inclu	ude/lwip 🔳			
SIZE: The mailbox siz	e for the topip threa		if(listIS_CONTAINED_WITHIN(NULL,	<pre>&(pxTimer->xTimerListItem)) == pdFALSE) /*lint !e961. The cast is only redundant when NULL is passed</pre>
P MBOX SIZE II def	ined DOXYGEN			
ABOX SIZE 0				ove it. */~
na n	n 22 én du da			mer->xlimerListitem } ;;-
inponencs/twip/port/es	spsz/metude 🖉			
SIZE: The mailbox siz	e for the topip threa			
ABOX_SIZE C	CONFIG_LWIP_TCPIP			
Ant idf rize			<pre>mtCOVERAGE TEST MARKER();-</pre>	

From the above figure, the problem lies in lw a5, 20(s0), s0 is 0x0, which corresponds to a NULL pointer. And then analyze the previous assembly lw s0, 24(sp), the corresponding code segment is pxTimer = xMessage.u.xTimerParameters.pxTimer;, at this point the question can be changed to why pxTimer is NULL.

You can further locate the problem by adding the following debug log:

```
if (pxTimer == NULL) {
    ets_printf("xMessageID: %d\n", xMessage.xMessageID);
}
```

The problem was finally found to be that the timer was not successfully created. The pointer pointing to the timer is NULL. If subsequent timer operations are performed at this time, a Load access fault (LoadProhibited) error will occur due to the null pointer.



Precautions Not all Load access fault (LoadProhibited) problems can be simply located. As shown in the figure below:
```
ELF file SHA256: a25f43e6908a36ed
Rebooting...
Guru Meditation Error: Core 1 panic'ed (LoadProhibited). Exception was unhandled.
Core 1 register dump:
                            : 0x00060533 A0 : 0x80376ae6 A1
PC
      : 0x400358e6 PS
                                                                         : 0x3fca2590
0x400358e6: rom_i2c_writeReg_Mask in ROM
        : 0x0000006d A3
                                                                          : 0x0000004
A2
                             : 0x00000001 A4
                                                   : 0x0000004 A5
                             : 0x0000001d A8
A6
       : 0x0000000 A7
                                                  : 0x3fcef3d4 A9
                                                                         : 0x00000000
                                                  : 0x00060524 A9 : 0x00000001
      : 0x00060523 A11
                            : 0x0000004 A12
A10
A14 : 0x00000001 A15
EXCVADDR: 0x00000190 LBEG
                             : 0x3fc98824 SAR : 0x000000a EXCCAUSE: 0x0000001c
: 0x40056f5c LEND : 0x40056f72 LCOUNT : 0x0000000
0x40056f5c: memcpy in ROM
0x40056f72: memcpy in ROM
Backtrace: 0x400358e3:0x3fca2590 |<-CORRUPTED
0x400358e3: rom i2c writeReg Mask in ROM
```

The PC address points to the ROM function rom_i2c_writeReg_mask function, and the Backtrace does not have enough valid information. In this case, a satble reproduction is required for troubleshooting. If the problem is difficult to be reproduced, it may be caused by hardware or wild pointer.

Store access fault (StoreProhibited)

Test Case When an application tries to write to an invalid memory location, this type of CPU exception occurs. The test code is shown below:

```
typedef struct {
    uint8_t buf[10];
    uint8_t data;
} store_test_t;
void store_access_fault(store_test_t * store_test)
{
    store_test->data = 0x5F;
    printf("Data: %d\n", store_test->data);
}
void app_main(void)
{
    store_test_t * store_test = NULL;
    store_access_fault(store_test);
}
```

Exception Phenomenon

• RISC-V

In the RISC-V architecture chip, you can see the following exception printout. Analyze the assembly instructions (the steps can be referred to *Appendix 1 - ESP Disassembly Corresponding Instructions*), when this exception occurs, the PC address points to sb/sw and other store-related instructions. At this time, the exception value 0×0000000 a stored in MTVAL indicates that it wants to access the content in an array or structure starting from NULL address.

```
Guru Meditation Error: Core 0 panic'ed (Store access fault). Exception was.

→unhandled.

Core 0 register dump:
```

					(continued fr	om previous page)
Stack dump detected						
MEPC : 0x42006960	RA	: 0x42006984	SP	: 0x3fc8f7c0	GP	:
⇔0x3fc8b400						
TP : 0x3fc87d7c	ΤO	: 0x4005890e	Т1	: 0x2000000	Т2	:_
↔0x0000000						
S0/FP : 0x3c023000	S1	: 0x0000000	AO	: 0x0000000	A1	:_
⊶0x3fc8f3f8						
A2 : 0x0000000	A3	: 0x0000001	A4	: 0x3fc8c000	A5	:_
⇔0x000005f						
A6 : 0x60023000	A7	: 0x000000a	S2	: 0x0000000	s3	:_
⇔0x0000000						
S4 : 0x0000000	S5	: 0x0000000	S6	: 0x0000000	S7	:_
↔0x0000000						
S8 : 0x0000000	S9	: 0x0000000	S10	: 0x00000000	S11	:
\leftrightarrow 0 x 0 0 0 0 0 0 0 0						
T3 : 0x0000000	Τ4	: 0x0000000	Т5	: 0x00000000	Т6	:
\leftrightarrow 0 x 0 0 0 0 0 0 0 0						
MSTATUS : 0x00001881	MTVEC	: 0x40380001	MCAUSE	: 0x0000007	MTVAL	:_
⊶0x000000a						
⇔0x000000a						

• XTENSA

In the XTENSA architecture chip, you can see the following exception printout. Analyze the assembly instructions (steps can be referred to *Appendix 1 - ESP Disassembly Corresponding Instructions*), when this exception occurs, the PC address points to s8i/s32i and other store-related instructions. At this time, the exception value $0 \times 0000000a$ stored in EXCVADDR indicates that it wants to access the content in an array or structure starting from NULL address.

Guru M	editation Error	: Core	0 panic'ed (StoreProh	ibited). Except	ion was unhandled.	
Core	0 register dump	:					
PC	: 0x42007c21	PS	: 0x0006063	0 A0	: 0x82007c38	A1 :	
⇔0x3f	c97d70						
A2	: 0x0000000	A3	: 0x3c023f4	8 A4	: 0x3c023fc8	A5 :	
⇔0x3f	c97d90						
A6	: 0x3fc97d70	A7	: 0x0000000	c A8	: 0x000005f	A9 :_	
⊶0x3f	c97d00						
A10	: 0x0000031	A11	: 0x3fc97fa	c A12	: 0x3fc97d70	A13 :	
⊶0x00	00000c						
A14	: 0x3fc941e4	A15	: 0x0000000	0 SAR	: 0x0000004	EXCCAUSE:	
EXCVAD	DR: 0x0000000a	LBEG	: 0x400556d	5 LEND	: 0x400556e5	LCOUNT :	
⇔0xff	ffffd	2220	• • • • • • • • • • • • • • • • • • • •		• • • • • • • • • • • • • • • • • • • •	1000000	
Backtrace: 0x42007c1e:0x3fc97d70 0x42007c35:0x3fc97d90 0x4201923f:0x3fc97db0_ →0x40379e71:0x3fc97de0							

Location Method The location method is similar to *Load access fault* (*LoadProhibited*). Analyze the assembly code (steps can be referred to *Appendix 1 - ESP Disassembly Corresponding Instructions*), you will see the following instruction:

sb	a5,10(a0)	// RSIC-V
s8i	a8, a2, 10	// XTENSA

When this type of exception being triggered, MTVAL/EXCVADDR will automatically update to data related to the exception, as can be seen from the RISC-V exception information MTVAL : 0x0000000a. Usually, MTVAL being NULL means that the CPU is trying to write data to a NULL address, and close to NULL means trying to write data to a member of an array or structure starting from a NULL address.

• For the RSIC-V architecture, the PC and RA registers contain the function pointers of the exception, and the

corresponding function can be found from the elf

• For the Xtensa architecture, a backtrace will be printed during an exception, and the corresponding function can also be found from the elf

Actual Case Please refer to Test Case.

Notes Similar to the *Load access fault*(*LoadProhibited*) issue, there are wild pointers cases that cannot be analyzed, and further analysis is required based on the actual application code.

Instruction Access Fault(InstrFetchProhibited)

Test Case When an application tries to read instructions from an invalid address, this type of CPU exception occurs, and the PC register often points to an invalid memory address. The test code is as follows:

```
typedef void (*fptr_t) (void);
volatile fptr_t fptr = (fptr_t) 0x4;
fptr();
```

The corresponding RISC-V assembly instruction (steps can refer to *Appendix 1 - ESP Disassembly Corresponding Instructions*) is as follows:

Exception Phenomenon

• RISC-V

Guru Meditation Error →unhandled.	: Core	0 panic'ed (Ir	nstructio	n access fault). Except	ion was_
Core 0 register dump	:					
MEPC : 0x0000004	RA	: 0x42006964	SP	: 0x3fc8f7b0	GP	•
↔0x3fc8b400	101	• 0112000901	01	• • • • • • • • • • • • • • • • • • • •	01	• •
TP : 0x3fc87d8c	ΤO	: 0x4005890e	Т1	: 0x2000000	Т2	:_
$\hookrightarrow 0 \times 0 0 0 0 0 0 0 0$						
S0/FP : 0x3c023000	S1	: 0x00000000	AO	: 0x0000031	A1	:_
⇔0x3fc8f3f8						
A2 : 0x0000000	A3	: 0x0000001	A4	: 0x3fc8c000	A5	:_
\hookrightarrow 0 x 0 0 0 0 0 0 4						
A6 : 0x60023000	A7	: 0x000000a	S2	: 0x0000000	S3	:_
↔0x0000000						
S4 : 0x0000000	S5	: 0x00000000	S6	: 0x0000000	S7	:_
↔0x0000000						
S8 : 0x0000000	S9	: 0x00000000	S10	: 0x0000000	S11	:_
\hookrightarrow 0 x 0 0 0 0 0 0 0 0						
T3 : 0x0000000	Τ4	: 0x0000000	Τ5	: 0x0000000	Τ6	:
\hookrightarrow 0 x 0 0 0 0 0 0 0 0						
MSTATUS : 0x00001881	MTVEC	: 0x40380001	MCAUSE	: 0x0000001	MTVAL	:_
\hookrightarrow 0 x 0 0 0 0 0 0 0 4						

•	Xtensa
---	--------

Guru Me ⇔unhan	ditation Error dled.	Core () p	anic'ed (Ins	strFetchP	rc,	bhibited). Ex	ception	was
Core 0	register dump	:							
PC ⇔0x3ff	: 0x00000004 b4a30	ΡS	:	0x00060630	A0	:	0x800d539a	A1	: _
A2 ⇔0x3ff	: 0x400de738 b4a60	A3	:	0x3f40379c	A4	:	0x3f40381c	A5	:
A6 ⇔0x3ff	: 0x3ffb4a40 b49d0	A7	:	0x000000c	A8	:	0x800e4dbe	A9	:
A10 ↔0x000	: 0x00000031 0000c	A11	:	0x3ffafc64	A12	:	0x3ffb4a40	A13	:
A14 ↔0x000	: 0x3ffb2770 00014	A15	:	0x0000cdcd	SAR	:	0x0000004	EXCCAUSE	:-
EXCVADD ⇔0xfff	R: 0x00000004 ffffd	LBEG	:	0x400014fd	LEND	:	0x4000150d	LCOUNT	:-
Backtra ⇔0x400	ce: 0x00000001: 858e9:0x3ffb4ak	:0x3ffb4a o0	130	0x400d5397:	0x3ffb4a	160) 0x400e54f8:	0x3ffb4a	80_

Troubleshooting Method Usually, such problems are caused by wild pointers. In this case, the PC register has no value, and whether other registers are worth analyzing depends on the running time of the wild pointer. For this test code, the current function has triggered a CPU exception, so the RA register contains the pointer of the upper-layer function (Xtensa architecture using Backtrace), and the exception can be analyzed at this case. But most of the time, the RA register will be destroyed. You can follow the steps below to locate the problem:

- 1. Stably reproduce the problem, ensure that the problem is the same at each time
- 2. Find the pattern of the problem, such as what operations are performed, or which logs are printed before the exception starts
- 3. Analyze in conjunction with the code based on the location of the program exception, add debugging logs and gradually reducing the amount of project code, and see if the problem can still be reproduced.

Actual Case As shown in the figure below, the scene of the exception has been completely destroyed. There is no way to infer the cause of the problem based on the information. However, most registers are destroyed into a fixed value, which is also a pattern. In cases where all registers are destroyed, it is usually a problem caused by memory trampling, which can be analyzed using the heap memory debugging method.

Anim->Dir 1									
Guru Meditation Error: Core 0 panic'ed (InstrFetchProhibited). Exception was unhandled.									
Core Ø register dump:									
PC : 0x08a408a4	PS :	0x00050033	AØ :	0x08a408a4	A1 :	0x3fcada60			
A2 : 0x08a408a4	A3 :	0x08a408a4	A4 :	0x08a408a4	A5 :	0x08a408a4			
A6 : 0x08a408a4	A7 :	0x08a408a4	A8 :	0x08a408a4	A9 :	0x08a408a4			
A10 : 0x08a408a4	A11 :	0x08a408a4	A12	0x08a408a4	A13 :	0x08a408a4			
A14 : 0x08a408a4	A15 :	0x08a408a4	SAR	0x00000024	EXCCAUSE:	0x00000014			
EXCVADDR: 0x08a408a4	LBEG :	0x08a408a4	LEND	0x08a408a4	LCOUNT :	0x08a408a4			
Backtrace: 0x08a408a1:	0x3fcada6	0 0x08a408a1	:0x08a408a	4 <- CORRUPT	ED				
ELF file SHA256: 73673d3b6									

Illegal Instruction

Test Case During the CPU instruction fetching phase, an illegal instruction (defined in the RISC-V spec as a 16-bit instruction all set to 0) will print this error. The test code is as follows:

Note: The ESP32 in the Xtensa platform does not support the above test case. If other chips want to run this test case, they need to turn off the memory protection function first.

Exception Phenomenon

• RISC-V

Guru Med →unhand	itation Error: led.	Core 0	panic'ed (I	llegal in	struction). Exc	eption was_	
Core 0 stack dur	register dump: mp detected						
MEPC ⇔0x3fc8b	: 0x4005f002 b400	RA	: 0x42006962	SP	: 0x3fc8f7c0	GP :	
0x4200690 ⇔access_	62: illegal_ir _fault/main/he	nstr_fault ello_world	t at /home/j d_main.c:79	acques/us	eful_example/c	ash_sim/store	_
TP ↔0x00000	: 0x3fc87d8c 0000	ТО	: 0x4005890e	Τ1	: 0x2000000	T2 :	
SO/FP ⇔0x3fc8i	: 0x3c023000 f3f8	S1	: 0x0000000	AO	: 0x3fc8f840	A1 :	
A2 ↔0x4005:	: 0x00000000 £000	A3	: 0x0000001	A4	: 0x3fc8c000	A5 :	
A6 ↔0x00000	: 0x60023000 0000	A7	: 0x000000a	S2	: 0x00000000	S3 :	
S4 ⇔0x00000	: 0x00000000 0000	S5	: 0x0000000	S6	: 0x00000000	s7 :	
S8 ⇔0x00000	: 0x00000000 0000	S9	: 0x0000000	S10	: 0x0000000	S11 :	
T3 ↔0x00000	: 0x00000000 0000	Τ4	: 0x0000000	Τ5	: 0x0000000	T6 :_	
MSTATUS ⇔0x00000	: 0x00001881 0000	MTVEC	: 0x40380001	MCAUSE	: 0x0000002	MTVAL :_	

• Xtensa

Guru Meditation Error: →unhandled.	Core 0	panic'ed (Ill	.egalInsti	ruction). Exce	eption was_
Memory dump at 0x403e2	98c: 0000	00000 000000000000000000000000000000000	00000000)	
Core 0 register dump:					
PC : 0x403e2990	PS	: 0x00060130	A0 :	0x820059fe	A1 :
⇔0x3fcf2990					
A2 : 0x4200c030	A3	: 0x3c0232fc	A4 :	0x3c02337c	A5 :
⇔0x3fcf29c0					
A6 : 0x3fcf29a0	A7	: 0x000000c	A8 :	0x820059f4	A9 :
⇔0x3fcf2930					
A10 : 0x0000031	A11	: 0x3fcf2be4	A12 :	0x3fcf29a0	A13 :
↔0x0000000c					(continues on next page)

Espressif Systems

					(continued from previous	page)	
A14 : 0x3fc922c4 →0x00000000	A15	: 0x0000000	SAR	: 0x0000004	EXCCAUSE:		
EXCVADDR: 0x0000000 →0xffffffd	LBEG	: 0x400556d5	LEND	: 0x400556e5	LCOUNT :		
Backtrace: 0x403e298d:0x3fcf2990 <-CORRUPTED							

Troubleshooting Although the ESP-IDF documentation contains Explanation of Illegal instruction, which mentions that there may be 3 types of situations that trigger this issue, in actual use, most of them are caused by the CPU fetching instructions from flash abnormally. Another part of them are caused by wild pointers fetching abnormal memory. From the PC address, it can be seen whether the addressing is abnormal from IRAM or flash.

Senarios for problems with fetching instructions from flash include: flash powering down when chip is sleeping, unstable CS pull-up, unstable flash power supply, and flash suspend function, etc.

Actual Case The following diagram shows an abnormal senario of the ESP chip in a low-power scenario. From the log, there is a power save printout before the error, which suggests that the problem may be related to the chip' s sleep mode. Later, by disabling the CONFIG turn off the flash power during sleep (CON-FIG_ESP_SLEEP_POWER_DOWN_FLASH) and enabling the CONFIG CS software pull-up during sleep (CON-FIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND), the problem no longer occurs.

cost steep partery_read_key penning battery_read_key:3 is_wifi_enable=0 is_standalone=0 test sleep UPM_init before sleep IPM init before (11323) pm: Frequency switching config: CPU_MAX: 160, APB_MAX: 80, APB_MIN: 10, Light sleep: ENABLED (11324) sleep: Enable automatic switching of GPIO sleep configuration test sleep UPM_init behind Guru Meditation Error: Core 0 panic'ed (Illegal instruction). Exception was unhandled. Stack dump detected Core 0 register dump: MEPC : 0x42002f7c : 0x42002f7c RA : 0x42004f82 SP : 0x3fca48e0 GP : 0x3fc94400 0x42002f7c: heap_caps_init at E:/utec/esp-idf0829/components/heap/heap_caps_init.c:94 (discriminator 1) 0x42004f82: app_main at E:/utec/lock/esp_lock/main/main.cpp:315 TP : 0x3fc7ef88 T0 : 0x4005890e T1 : 0x42001a3e T2 : 0xffffffff 0x42001a3e: console_write at E:/utec/esp-idf0829/components/vfs/vfs_console.c:73 : 0x3c0e5d8c A1 : 0x3c0e6000 A0 SO/FP : 0x3c0e6000 S1 : 0x3fca4958 : 0x00000001 A4 : 0x000000c8 S2 : 0x600c0000 A5 : 0x00000000 S3 : 0x000000x0 Α2 : 0x00000000 A3 : 0x00000010 A7 Α6 : 0x0007a000

Cache disable but cached memory region accessed (Cache error)

Abnormal Reason During flash operations (read/write/erase), the cache is turned off for a short time. If an interrupt is triggered during this time (interrupts decorated with ESP_INTR_FLAG_IRAM can be triggered when the cache is off), such as if there are functions in the interrupt function that need to fetch instructions or data from the flash through the cache (includingmmap), an exception will occur because the cache is turned off.

Test Case After turning off the cache, access and print the data in the flash. The test code is shown below:

```
static const uint32_t s_in_rodata[8] = { 0x12345678, 0xfedcba98 };
spi_flash_disable_interrupts_caches_and_other_cpu();
volatile uint32_t* src = (volatile uint32_t*) s_in_rodata;
uint32_t v1 = src[0];
uint32_t v2 = src[1];
ets_printf("%lx %lx\n", v1, v2);
```

Abnormal Phenomenon

• RISC-V

Guru Meditation Error: Core 0 panic'ed (Cache error). access to cache while dbus or cache is disabled							
Stack dump detected Core 0 register dump	:						
MEPC : 0x40000040 →0x3fc8b400	RA	: 0x40381e82	SP	: 0x3fc8f7c0	GP :		
0x40381e82: cache_acce →access_fault/main/he	ess_test_ ello_worl	_func at /home/ ld_main.c:38	/jacques/	useful_example	e/crash_sim/store_		
TP : 0x3fc87d5c →0x0000000	ТО	: 0x4005890e	Τ1	: 0x2000000	T2 :_		
SO/FP : 0x3c023000 →0x00000000	S1	: 0x0000000	AO	: 0x3c025610	A1 :_		
A2 : 0x0000000 →0x3c02561c	A3	: 0x0000200	A4	: 0x600c4000	A5 :_		
A6 : 0x60023000 →0x00000000	A7	: 0x000000a	S2	: 0x0000000	S3 :		
S4 : 0x0000000 →0x00000000	S5	: 0x0000000	S6	: 0x00000000	s7 :_		
S8 : 0x0000000 →0x0000000	S9	: 0x0000000	S10	: 0x0000000	S11 :_		
T3 : 0x0000000 →0x00000000	Τ4	: 0x0000000	Τ5	: 0x0000000	тб :_		
MSTATUS : 0x00001881 ⊶0x6944806f	MTVEC	: 0x40380001	MCAUSE	: 0x00000019	MTVAL :_		

• Xtensa

```
Guru Meditation Error: Core 0 panic'ed (Cache disabled but cached memory region_
\leftrightarrow accessed).
Core 0 register dump:
PC : 0x400830ed PS
                           : 0x00060a34 A0
                                                : 0x800d539a A1
                                                                      ÷. .
→0x3ffb4a40
0x400830ed: cache_access_test_func at /home/jacques/useful_example/crash_sim/store_
→access_fault/main/hello_world_main.c:32
A2
      : 0x400de738 A3 : 0x3f40379c A4
                                                : 0x3f40381c A5
                                                                      \leftrightarrow 0x3ffb4a60
0x400de738: vprintf at /builds/idf/crosstool-NG/.build/xtensa-esp32-elf/src/newlib/
→newlib/libc/stdio/vprintf.c:30
   : 0x3ffb4a40 A7 : 0x000000c A8
                                                 : 0x3f4040c4 A9
A6
                                                                      ÷. .
→0x3ffb4a10
A10 : 0x0000001 A11 : 0xbad00bad A12
                                                 : 0x3ffb2608 A13
                                                                      →0x000000c
A14 : 0x3ffb2770 A15
                           : 0x0000cdcd SAR
                                                 : 0x0000004 EXCCAUSE:__
-0x00000007
EXCVADDR: 0x00000000 LBEG : 0x400014fd LEND
                                                 : 0x4000150d LCOUNT :_
⇔0xffffffd
Backtrace: 0x400830ea:0x3ffb4a40 0x400d5397:0x3ffb4a60 0x400e54e4:0x3ffb4a80_
→0x40085915:0x3ffb4ab0
```

Analysis Method In such cases, focus on the PC address of the exception. If the address is in flash, it means that instructions are read from flash when the cache is closed. You need to decorate the corresponding function with IRAM_ATTR. It should be noted that IRAM_ATTR is only for the current function. If this function has sub-functions,

all the sub-functions need to be decorated with IRAM_ATTR.

If the address is in IRAM, it means that there is no problem with this function itself, but the input parameters of this function may need to be read from flash.

Practical Case During flash read and write operations (causing cache close), the gptimer interrupt that has been put into IRAM is also triggered. The gptimer ISR executes some operations that need to read content from flash, triggering this problem:



From the figure above, we can conclude several problem points

- 1. It can be confirmed that the gptimer interrupt is triggered during flash read and write operations, and there is content in the flash in the ISR
- 2. The function gpio_set_level pointed by the PC has been put into IRAM instead of flash, so it should run normally when the cache is closed

After analysis, it was finally found that the problem was caused by the input parameter LED_matrix of the function gpio_set_level being decorated with const. The parameters decorated with const will be placed in the .rodata section of flash, so reading this parameter requires cache.

```
//static bool LEDsTimer on alarm(gptimer handle t LEDtimer hdl, const gptimer alarm e
static bool IRAM ATTR LEDsTimer on alarm(gptimer handle t LEDtimer hdl, const gptimer
   int64 t start, diff;
   start = esp timer get time();
   BaseType t high task awoken = pdFALSE;
   uint8 t count;
    static bool pin level = 0;
    gpio set level( IOO PIN, 1 );
   ledc timer rst( ledc timer.speed mode, ledc timer.timer num ); //To synchronized
#if 1
    for ( count = 0; count < GPIO NUM MAX ROW; count++ )
        gpio_set_level( LED_matrix.row_GPI0[count], 0 );
    }
    for ( count = 0; count < GPIO NUM MAX COL; count++ )
        //uint8 t index = row count * GPIO NUM MAX COL + count;
        /***** Not SAFE
        ledc set duty(ledc channel[count].speed_mode, ledc_channel[count].channel, dut
        ledc update duty(ledc channel[count].speed mode, ledc channel[count].channel);
                      and update(ledc_channel[count]
```

Precautions The exception of Cache disable is better captured on the Xtensa platform, and may appear as other errors caused by the inability to read data from the cache on the RISC-V platform, such as Illegal instruction. The code example is as follows:

```
gpio_set_level(12, 1);
spi_flash_disable_interrupts_caches_and_other_cpu();
gpio_set_level(12, 0);
```

When running in ESP32-C3 (RISC-V platform), the crash is as follows

```
Guru Meditation Error: Core 0 panic'ed (Illegal instruction). Exception was_
\hookrightarrowunhandled.
Core 0 register dump:
Stack dump detected
MEPC : 0x42006ef2 RA : 0x40381e7e SP : 0x3fc8f7f0 GP
                                                                   . . .
⇔0x3fc8b400
0x42006ef2: gpio_set_level at /home/jacques/sdk/esp-idf/components/driver/gpio/
⇔gpio.c:233
0x40381e7e: cache_access_test_func at /home/jacques/useful_example/crash_test/main/
→hello_world_main.c:50
    : 0x3fc87d4c T0
                          : 0x4005890e T1
                                               : 0x20000000 T2
ΤP
                                                                    . ....
→0x00000000
S0/FP : 0x3c023000 S1
                          : 0x0000000 A0
                                               : 0x000000c A1
                                                                    : _
⇔0x00000000
A2 : 0x0000200 A3
                          : 0x00000200 A4
                                                : 0x600c4000 A5
                                                                     : __
```

						(continued in	om previous page)
A6	: 0x60023000	А7	: 0x000000a	S2	: 0x00000000	S3	:_
→ 0x000	00000						
S4	: 0x00000000	S5	: 0x0000000	S6	: 0x00000000	S7	:_
<u></u> →0x000	00000						
S8	: 0x0000000	S9	: 0x0000000	S10	: 0x00000000	S11	:_
<u></u> →0x000	00000						
ТЗ	: 0x00000000	Τ4	: 0x0000000	Т5	: 0x00000000	Тб	:_
<u></u> →0x000	00000						
MSTATUS	: 0x00001881	MTVEC	: 0x40380001	MCAUSE	: 0x0000002	MTVAL	:_
⊶0x000	00000						

Interrupt wdt timeout on CPU0/CPU1

Exception Principle The interrupt watchdog uses the hardware watchdog timer on Timer group 1 to monitor system task scheduling by incorporating a dog-feeding operation in the SysTick interrupt. If the dog-feeding operation is not executed within the SysTick interrupt for an extended period (default 300 ms), the interrupt watchdog interrupt is triggered. For more information, please refer to Interrupt Watchdog Timer (IWDT).

Test Case This issue can be triggered by disabling the interrupt. The test code is as follows:

Exception Phenomenon

• RISC-V

```
Guru Meditation Error: Core 0 panic'ed (Interrupt wdt timeout on CPU0).
Core 0 register dump:
Stack dump detected
MEPC : 0x42006964 RA : 0x42006964 SP : 0x3fc911a0 GP
                                                           . .
⇔0x3fc8b400
0x42006964: interrupt_wdt_fault_task at /home/jacques/useful_example/crash_test/
→main/hello_world_main.c:121 (discriminator 1)
TP : 0x3fc8971c TO : 0x0000000 T1
                                           : 0x0000000 T2
                                                               ÷...
↔0x00000000
SO/FP : 0x0000000 S1 : 0x0000000 A0
                                            : 0x0000001 A1
                                                                ÷. .
⇔0x0000000
A2 : 0x0000000 A3 : 0x0000004 A4
                                            : 0x0000001 A5
                                                                ÷. .
→0x3fc8c000
A6 : 0x0000000 A7
                        : 0x0000000 S2
                                            : 0x0000000 S3
                                                                . ...
⇔0x0000000
S4 : 0x0000000 S5
                        : 0x0000000 S6
                                             : 0x0000000 S7
                                                                : ___
↔0x0000000
```

S8 : 0x	000000000	S 9	:	0x00000000	S10	:	0x00000000	S11	:_
↔0x0000000)								
T3 : 0x	000000000	Т4	:	0x00000000	Т5	:	0x00000000	Т6	:_
⇔0x0000000)								
MSTATUS : 0x	00001881	MTVEC	:	0x40380001	MCAUSE	:	0x0000018	MTVAL	:_
⇔0x0000a001	-								

• Xtensa

Guru Meditation Error: Core 0 panic'ed (Interrupt wdt timeout on CPU0). Core 0 register dump: РC : 0x40378b06 PS : 0x00060634 A0 : 0x82002239 A1 <u>ب</u> →0x3fc984f0 0x40378b06: esp_cpu_wait_for_intr at /home/jacques/sdk/esp-idf/components/esp_hw_ ⇔support/cpu.c:110 : 0x0000000 A3 : 0x0000000 A4 : 0x3fc96c70 A5 A2 →0x3fc96c50 : 0x0000000 A8 : 0x42005e10 A7 : 0x82009452 A9 A6 <u>۔</u> $\rightarrow 0x3fc984b0$ 0x42005e10: timer_task at /home/jacques/sdk/esp-idf/components/esp_timer/src/esp_ →timer.c:470 A10 : 0x0000000 A11 : 0x0000000 A12 : 0x3fc96c50 A13 <u>۔</u> ⇔0x3fc96c30 A14 : 0x0000000 A15 : 0x0000000 SAR : 0x0000000 EXCCAUSE: ↔0x0000005 EXCVADDR: 0x00000000 LBEG : 0x00000000 LEND : 0x0000000 LCOUNT :_ Backtrace: 0x40378b03:0x3fc984f0 0x42002236:0x3fc98510 0x4037b5a9:0x3fc98530_ ↔0x40379e71:0x3fc98550 0x40378b03: xt_utils_wait_for_intr at /home/jacques/sdk/esp-idf/components/xtensa/ →include/xt_utils.h:81 (inlined by) esp_cpu_wait_for_intr at /home/jacques/sdk/esp-idf/components/esp_hw_ →support/cpu.c:101 0x42002236: esp_vApplicationIdleHook at /home/jacques/sdk/esp-idf/components/esp_ →system/freertos_hooks.c:59 0x4037b5a9: prvIdleTask at /home/jacques/sdk/esp-idf/components/freertos/FreeRTOS-→Kernel/tasks.c:4269 (discriminator 1) 0x40379e71: vPortTaskWrapper at /home/jacques/sdk/esp-idf/components/freertos/ →FreeRTOS-Kernel/portable/xtensa/port.c:149 Core 1 register dump: : 0x42007c32 PS : 0x00060034 A0 : 0x80379e74 A1 PC · _ \leftrightarrow 0x3fc99f20 0x42007c32: interrupt_wdt_fault_task at /home/jacques/useful_example/crash_test/ →main/hello_world_main.c:121 (discriminator 1) : 0x0000001 A5 : 0x0000000 A3 : 0x0000000 A4 A2 <u>۔</u> ↔0x0000000 A6 : 0x0000001 A7 : 0x0000000 A8 : 0x82007c32 A9 <u>۔</u> ⇔0x3fc99ef0 : 0x0000001 A11 : 0x3fc93a54 A12 : 0x00060020 A13 A10 <u>۔</u> ↔0x00060023 : 0x00060323 A15 : 0x0000abab SAR : 0x0000000 EXCCAUSE:_ A14 \rightarrow 0x00000005 EXCVADDR: 0x00000000 LBEG : 0x00000000 LEND : 0x0000000 LCOUNT :__ ↔0x00000000

```
Backtrace: 0x42007c2f:0x3fc99f20 0x40379e71:0x3fc99f40
0x42007c2f: vPortEnterCritical at /home/jacques/sdk/esp-idf/components/freertos/
→FreeRTOS-Kernel/portable/xtensa/include/freertos/portmacro.h:573
(inlined by) interrupt_wdt_fault_task at /home/jacques/useful_example/crash_test/
→main/hello_world_main.c:120
0x40379e71: vPortTaskWrapper at /home/jacques/sdk/esp-idf/components/freertos/
→FreeRTOS-Kernel/portable/xtensa/port.c:149
```

Analysis Method In case of such problems, primarily observe the PC address of the exception to determine the location of the crash trigger. For multi-core architectures, it is necessary to observe the registers of several cores and analyze the methods that trigger the problem.

The possible reasons and specific explanations are as follows:

- 1. Long time interrupt off
- 2. Blocking issue exists in ISR function
- 3. Interrupt not cleared

Long time interrupt off Turning off the interrupt is designed to protect critical code sections from being interrupted. However, long time interrupt off is a primary cause of triggering the interrupt watchdog.

As in *Test Case*, operations like entering a critical section, the interrupt is turned off, preventing the SysTick interrupt from executing the watchdog feeding operation in time. This leads to an interrupt watchdog timeout and triggers the watchdog exception handler. Users must troubleshoot the logic of interrupt disabling in the problematic code area.

Blocking in ISR function When an interrupt occurs, the corresponding ISR is called for processing. ISR functions should perform minimal operations, deferring time-consuming tasks to non-interrupt functions. If an ISR takes too long to execute or blocks, it may trigger the interrupt watchdog.

Here is a simple example of GPIO ISR blocking:

```
#define GPIO_INPUT_IO_0
                              4
static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    ets_printf("ISR\n");
    while(1){}
}
void app_main(void)
{
    gpio_config_t io_conf;
    io_conf.intr_type = GPIO_INTR_POSEDGE;
    io_conf.pin_bit_mask = (1ULL<<GPIO_INPUT_IO_0);</pre>
    io_conf.mode = GPIO_MODE_INPUT;
    io_conf.pull_up_en = 1;
    gpio_config(&io_conf);
    gpio_install_isr_service(0);
    gpio_isr_handler_add(GPIO_INPUT_IO_0, gpio_isr_handler, (void*) GPIO_INPUT_IO_
\rightarrow 0);
}
```

Manually pull up GPIO 4 to trigger the GPIO interrupt, and print the following crash information:

Chapter 2. ESP Friends

```
ISR
Guru Meditation Error: Core 0 panic'ed (Interrupt wdt timeout on CPU0).
Core 0 register dump:
PC
     : 0x40082580 PS : 0x00060034 A0
                                                 : 0x80081560 A1
                                                                        : ...
⇔0x3ffb07c0
0x40082580: gpio_isr_handler at /home/jacques/problem/wdt/int_wdt/build/../main/
→hello_world_main.c:40 (discriminator 1)
A2
      : 0x00000004 A3 : 0x3ff0015c A4
                                                 : 0x800d6d82 A5
                                                                        <u>۔</u>
⇔0x3ffb3400
A6 : 0x0000000 A7 : 0x0000001 A8
                                                  : 0x80082580 A9
                                                                        •
\leftrightarrow 0x3ffb0730
A10 : 0x0000004 A11
                           : 0x0000004 A12
                                                  : 0x0000004 A13
                                                                        ÷. .
⇔0x0000000
A14 : 0x0000000 A15
                           : 0x3ffb3400 SAR
                                                  : 0x000001c EXCCAUSE:
\rightarrow 0 \times 0 0 0 0 0 0 0 5
EXCVADDR: 0x00000000 LBEG : 0x00000000 LEND
                                                 : 0x00000000 LCOUNT :_
\rightarrow 0 \times 0 0 0 0 0 0 0 0
Core 0 was running in ISR context:
EPC1 : 0x400d1097 EPC2 : 0x0000000 EPC3
                                                 : 0x0000000 EPC4
                                                                        ÷. .
↔0x40082580
0x400d1097: uart_hal_write_txfifo at /home/jacques/sdk/esp-idf/components/hal/uart_
→hal_iram.c:35
0x40082580: gpio_isr_handler at /home/jacques/problem/wdt/int_wdt/build/../main/
→hello_world_main.c:40 (discriminator 1)
Backtrace:0x4008257d:0x3ffb07c0 0x4008155d:0x3ffb07e0 0x40082102:0x3ffb0800_
↔0x400d6702:0x3ffb3450 0x400862a2:0x3ffb3470 0x40087865:0x3ffb3490
```

Through the PC and Backtrace, we can confirm that the problem occurs in the gpio_isr_handle function. In addition, the log line Core 0 was running in ISR context also indicates that it is currently running in the interrupt function.

Interrupt not cleared After the ISR function ends, the corresponding interrupt must be manually cleared. If it is not cleared at this time, the interrupt will keep triggering, preventing the SysTick interrupt from responding.

As in the example *Blocking in ISR function*, if you remove the while(1) loop in gpio_isr_handler, and comment out gpio_hal_clear_intr_status_bit in the ESP-IDF GPIO driver, recompiling and running the program will still trigger the interrupt watchdog crash.

If the user has not modified the default driver, interrupt clearing is usually handled by the lower-layer driver. If an interrupt watchdog exception points to the internal driver, check whether the lower-layer driver properly clears the interrupt.

Real Case Example 1: Interrupt Watchdog Caused by FreeRTOS System Exception

In practical applications, interrupt watchdog issues often do not exist independently, they are usually combined with FreeRTOS. Many operations in FreeRTOS require interrupt handling, and if FreeRTOS encounters an exception, it is prone to freeze and manifest as an interrupt watchdog issue.

The actual crash case example is as follows:

```
Guru Meditation Error: Core 0 panic'ed (Interrupt wdt timeout on CPU0)

Core 0 register dump:

PC : 0x400915d2 PS : 0x00060b34 A0 : 0x800903ed A1 :..

→0x3ffd4e80

A2 : 0x3ffd3598 A3 : 0x3ffd4ff0 A4 : 0x00000c9f A5 :..

→0x3f408990

A6 : 0x3ffbf580 A7 : 0x00060023 A8 : 0x3ffd4ff0 A9 :..

→0x0000000f
```

```
(continued from previous page)
```

```
A10
      : 0x3ffd4ff0 A11 : 0x0000000f A12
                                               : 0x00060b20 A13
                                                                    →0x0000001
     : 0x3ffeddf0 A15
                          : 0x3ffea57c SAR
                                               : 0x00000018 EXCCAUSE:
A14
↔0x0000005
EXCVADDR: 0x00000000 LBEG : 0x400014fd LEND
                                               : 0x4000150d LCOUNT :_
⇔0xffffffe
ELF file SHA256: eadf38922e335ed1
Backtrace: 0x400915cf:0x3ffd4e80 0x400903ea:0x3ffd4ea0 0x4008f514:0x3ffd4ec0_
→0x400ffb91:0x3ffd4f00 0x40136612:0x3ffd4f30
```

Analyzing the Backtrace information, as follows:

```
~$ xtensa-esp32-elf-addr2line -afe esp32_interrupt_wdt.elf 0x400915cf:0x3ffd4e80_
→0x400903ea:0x3ffd4ea0 0x4008f514:0x3ffd4ec0 0x400ffb91:0x3ffd4f00_
→0x40136612:0x3ffd4f30
0x400915cf
vListInsert
/arm/esp-idf/components/freertos/list.c:205
0x400903ea
vTaskPlaceOnEventList
/arm/esp-idf/components/freertos/tasks.c:2901 (discriminator 2)
0x4008f514
xQueueGenericReceive
/arm/esp-idf/components/freertos/queue.c:1596
0x400ffb91
arch_os_mbox_fetch
/arm/components/esp32/arch_os.c:352 (discriminator 4)
0x40136612
msg_fetch_block
/arm/components/libs/d0/delayzero.c:631
```

From the above log, the problem is not a crash in the interrupt, but points to FreeRTOS itself. By searching for vListInsert, it shows that the problem occurs in the following for loop:

```
if( xValueOfInsertion == portMAX_DELAY )
{
    pxIterator = pxList->xListEnd.pxPrevious;
}
else
{
    for( pxIterator = ( ListItem_t * ) &( pxList->xListEnd ); pxIterator->pxNext->
    ixItemValue <= xValueOfInsertion; pxIterator = pxIterator->pxNext )
    {
        /* There is nothing to do here, just iterating to the wanted
        insertion position. */
    }
}
```

Further check whether the corresponding upper-layer code has interrupt operations at this moment. Finally in vTaskPlaceOnEventList, it shows that interrupts are indeed disabled before performing the list insertion operation.

```
(continues on next page)
```

```
prvAddCurrentTaskToDelayedList( xPortGetCoreID(), xTicksToWait);
taskEXIT_CRITICAL(&xTaskQueueMutex);
```

}

For now the analysis is complete. This issue is a FreeRTOS problem that manifests as an interrupt watchdog. To troubleshoot this type of issue, we need to analyze why it no longer exits in the for loop. It is unlikely that the issue lies with FreeRTOS itself, rather, it is typically caused by improper use of the upper-layer application code, such as memory stomping, blocking operations within interrupts, or similar issues.

Analysis Techniques and Considerations

1. Quickly Determine the Problem Type

After encountering a watchdog issue, if the crash information includes Core X was running in ISR context, it essentially indicates that the problem lies within the interrupt function. Conversely, it suggests that the issue is due to the disabling of interrupts.

2. Complexity of System-Related Issues

In actual applications, the interrupt watchdog does not exist independently, it is generally combined with the system. Many operations in FreeRTOS require interrupt handling, and if the system encounters an exception, it can easily get stuck, manifesting as an interrupt watchdog issue.

- 3. Other Practical Example References
 - Interrupt watchdog not cleared
 - Blocking exists in ISR function

Appendix 1 - ESP Disassembly Corresponding Instructions

- Xtensa ESP32:xtensa-esp32-elf-objdump -S xxx.elf > a.txt
- Xtensa ESP32-S2:xtensa-esp32s2-elf-objdump -S xxx.elf > a.txt
- Xtensa ESP32-S3: xtensa-esp32s3-elf-objdump -S xxx.elf > a.txt
- RISC-V:riscv32-esp-elf-objdump -S xxx.elf > a.txt

Where xxx.elf is the project elf file, which is often in the build folder of the project directory. a.txt is the output of the disassembly, which contains the actual assembly instructions.

Locating Problems Using Backtrace & Coredump

Backtrace Overview By default, when a Panic occurs, unless *CONFIG_ESP_SYSTEM_PANIC_SILENT_REBOOT* is enabled, the panic handler will print CPU registers and backtrace to the console for developers to further debug and analyze. This is further described in the ESP-IDF Programming Guide in the section on Register Dump and Backtrace.

Backtrace information varies for different CPU architectures. Currently, ESP primarily uses the Xtensa architecture, exemplified by ESP32-S3, and the RISC-V architecture, exemplified by ESP32-C3:

- For Xtensa, the focus is on PC and EXCVADDR, as well as the Backtrace address.
- For RISC-V, the focus is on MEPC and MTVAL, as well as stack information (Backtrace address is not printed by default).

Register information and Backtrace addresses can be printed and manually parsed using third-party serial tools and *addr2line* or automatically through *idf.py monitor*. We will elaborate on how to use Backtrace & Coredump to locate issues for these two CPU architectures.

xTensa Backtrace ESP currently uses the xTensa architecture for chips such as ESP8266, ESP32, ESP32-S2, and ESP32-S3.

Here is an example of error logs corresponding to xTensa:

Guru Meditation Error: C	Core 0 panic'ed (Loa	adProhibite	ed). Exceptio	on was un	handled.
Core 0 register dump:					
PC : 0x42016d2f PS	: 0x00060c30	A0 :	0x820088ac	A1	:
⇔UX3IC98dUU					
A2 : 0x0000000 A3	3 : 0x3fc98f34	A4 :	0x0000000a	A5	:_
⇔0x3fc98d20					
A6 : 0x3fc98d00 A7	: 0x000000c	A8 :	0x8200ca00	A9	:
⇔0x3fc989b0					
A10 : 0x0000002 A1	1 : Oxfffffff	A12 :	0x0000002	A13	:_
⇔0x3fc98bc0					
A14 : 0x3fc989c0 A1	.5 : 0x0000001	SAR :	0x0000019	EXCCAUSE	:
⇔0x000001c					
EXCVADDR: 0x0000008 LB	BEG : 0x400556d5	LEND :	0x400556e5	LCOUNT	:
⇔0xffffff9					
Backtrace, 0x42016d2c.0x	2f-08d00 0x420088-9	• 0 v 3 f a 98 d 20) 0v/20180ff.	0v3fa98d	10
→0x4037a2dd:0x3fc98d70	SICJUUU 0X420000d9	. UAJICJOUZU	, 0747010011.	0701000	

You can further use the *addr2line* command to parse the Backtrace into readable function names based on the *.elf* file. An example command is as follows:

xtensa-esp32s3-elf-addr2line -pfiaC -e path.elf 0x42016d2c:0x3fc98d00

The corresponding result is:

If you use IDF monitor to print logs, it will automatically call the above *addr2line* and print the parsed results, as shown below:

Guru Medi	tation Error	: Core O	panic'ed (Lo	adProhibi	ted). Exceptio	on was unhandled.
Core 0 r	egister dump	:				
PC :	0x42016d2f d00	PS	: 0x00060c30	A0	: 0x820088ac	A1 :_
0x42016d2 ⇔debug_m	f: new_monkey ethod.c:19	y_born at	/home/libo/t	est_githu	ub/idf_debug_me	<pre>sthod/main/idf_</pre>
A2 : →0x3fc98	0x00000000 d20	A3	: 0x3fc98f34	A4	: 0x0000000a	A5 :_
A6 : ⇔0x3fc98	0x3fc98d00 9b0	А7	: 0x000000c	A8	: 0x8200ca00	A9 :_
A10 : →0x3fc981	0x00000002 bc0	A11	: Oxfffffff	A12	: 0x0000002	A13 :_
A14 : ⇔0x00000	0x3fc989c0 01c	A15	: 0x0000001	SAR	: 0x0000019	EXCCAUSE:
EXCVADDR: →0xffff	0x0000008 ff9	LBEG	: 0x400556d5	LEND	: 0x400556e5	LCOUNT :_
0x400556d	5: strlen in	ROM				

```
0x400556e5: strlen in ROM
```

```
Backtrace: 0x42016d2c:0x3fc98d00 0x420088a9:0x3fc98d20 0x420180ff:0x3fc98d40_

→0x4037a2dd:0x3fc98d70

0x42016d2c: new_monkey_born at /home/libo/test_github/idf_debug_method/main/idf_

→debug_method.c:18

0x420088a9: app_main at /home/libo/test_github/idf_debug_method/main/idf_debug_

→method.c:70

0x420180ff: main_task at /home/libo/esp/github_master/components/freertos/app_

→startup.c:208 (discriminator 13)

0x4037a2dd: vPortTaskWrapper at /home/libo/esp/github_master/components/freertos/

→FreeRTOS-Kernel/portable/xtensa/port.c:162
```

Now, based on the detailed backtrace log, Guru Meditation Error, and the pointers:

- PC: Program Counter
- EXCVADDR: Exception Vector Address

you can further debug and analyze the common causes of Guru Meditation errors, as explained in the section on *Locating Problems Using Guru Meditation Error Printing* in the documentation.

RISC-V Backtrace Currently, ESP uses RISC-V architecture for chips such as ESP32-C3, ESP32-C2, ESP32-C6, and ESP32-H2.

Here is an example of error logs corresponding to RISC-V:

```
Guru Meditation Error: Core 0 panic'ed (Load access fault). Exception was_
⇔unhandled.
Core 0 register dump:
MEPC : 0x42007988 RA : 0x42007a4e SP
                                               : 0x3fc8fed0 GP
                                                                    . ...
→0x3fc8b200
    : 0x3fc870f8 T0 : 0x4005890e T1
ТΡ
                                               : 0x3fc8fb2c T2
                                                                    : ___
⇔0x0000000
S0/FP : 0x000000a S1 : 0x3fc90948 A0
                                               : 0x0000000 A1
                                                                    • • •
⇔0x3fc8fb08
A2
      : 0x0000000 A3 : 0x0000001 A4
                                               : 0x0000000 A5
                                                                    ÷. .
↔0x0000009
A6 : 0x60023000 A7 : 0x000000a S2
                                               : 0x0000000 S3
                                                                    ÷. .
→0x0000000
S4 : 0x0000000 S5 : 0x0000000 S6
                                               : 0x0000000 S7
                                                                    ⇔0x0000000
S8 : 0x0000000 S9
                          : 0x0000000 S10
                                               : 0x0000000 S11
                                                                    ÷.,
→0x0000000
T3 : 0x0000000 T4
                          : 0x0000000 T5
                                               : 0x0000000 T6
                                                                    1....
↔0x0000000
MSTATUS : 0x00001881 MTVEC : 0x40380001 MCAUSE : 0x00000005 MTVAL
                                                                    :...
\rightarrow 0 \times 0 0 0 0 0 0 0 8
MHARTID : 0x0000000
Stack memory:
3fc8fed0: 0x3c021bd0 0x00000000 0x3c022000 0x42015de4 0x00000000 0x00001388_
→0x0000001 0x0000000
3fc8fef0: 0x00000000 0x00000000 0x0000000 0x40384538 0x00000000 0x00000000_
→0x0000000 0x0000000
```

You can further use the *addr2line* command to parse the Backtrace into readable function names based on the *.elf* file. An example command is as follows:

riscv32-esp-elf-addr2line -pfiaC -e path.elf 0x42007988 0x42007a4e

The corresponding result is:

If you use IDF monitor to print logs, it will automatically call the above *addr2line* and print the parsed results, as shown below:

```
Guru Meditation Error: Core 0 panic'ed (Load access fault). Exception was_
\rightarrowunhandled.
Stack dump detected
Core 0 register dump:
MEPC : 0x42007988 RA : 0x42007a4e SP : 0x3fc8fed0 GP
                                                                      ____
→0x3fc8b200
0x42007988: new_monkey_born at /home/libo/test_github/idf_debug_method/main/idf_
→debug_method.c:19
0x42007a4e: app_main at /home/libo/test_github/idf_debug_method/main/idf_debug_
→method.c:71
TР
       : 0x3fc870f8 T0
                           : 0x4005890e T1
                                                 : 0x3fc8fb2c T2
                                                                        <u>۔</u>
→0x00000000
0x4005890e: memset in ROM
S0/FP : 0x000000a S1
                            : 0x3fc90948 A0
                                                  : 0x0000000 A1
                                                                        · ....
↔0x3fc8fb08
       : 0x0000000 A3
A2
                            : 0x0000001 A4
                                                  : 0x0000000 A5
                                                                        ÷. .
→0x0000009
       : 0x60023000 A7
                           : 0x000000a S2
                                                  : 0x0000000 S3
A 6
                                                                        ÷...
→0x00000000
      : 0x0000000 S5
                           : 0x0000000 S6
                                                  : 0x0000000 S7
S4
                                                                        ÷. .
→0x00000000
S8 : 0x0000000 S9
                           : 0x0000000 S10
                                                  : 0x0000000 S11
                                                                        . ...
⇔0x0000000
тЗ
      : 0x0000000 T4 : 0x0000000 T5
                                                 : 0x0000000 T6
                                                                        : ___
\rightarrow 0 \times 0 0 0 0 0 0 0 0
MSTATUS : 0x00001881 MTVEC : 0x40380001 MCAUSE : 0x00000005 MTVAL :_
⇔0x0000008
0x40380001: _vector_table at ??:?
MHARTID : 0x0000000
Backtrace:
new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_github/idf_debug_method/
→main/idf_debug_method.c:19
19
           zoo->monkey++;
#0 new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_github/idf_debug_method/
→main/idf_debug_method.c:19
#1 0x42007a4e in app_main () at /home/libo/test_github/idf_debug_method/main/idf_
→debug_method.c:70
#2 0x42015de4 in main_task (args=<error reading variable: value has been_
-optimized out>) at /home/libo/esp/github_master/components/freertos/app_startup.
⇔c:208
#3 0x40384538 in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
→<optimized out>) at /home/libo/esp/github_master/components/freertos(commeRonOmerate)
→Kernel/portable/riscv/port.c:234
```

Now, you can further debug and analyze the common causes of Guru Meditation errors using the detailed backtrace log, Guru Meditation Error, and the pointers:

- MEPC: Machine Exception Program Counter
- MTVAL: Machine Trap Value

Further debugging and analysis can be conducted based on pointers, Guru Meditation errors, and common causes of Guru Meditation errors, as explained in the section on *Locating Problems Using Guru Meditation Error Printing*.

Common Backtrace Errors Common Backtrace errors are summarized in the table below. Detailed information is also provided in the section on *Locating Problems Using Guru Meditation Error Printing*.

Xtensa	RISC-V	Why	Where
IllegalInstruction	IllegalInstruction	SPI Flash IO Broken / task re-	Hardware or Soft-
		turn without vTaskDelete / non-	ware
		void function return void	
•	Instruction Address Mis-	not 2-byte aligned	PC(0x4),
	aligned		MEPC(0x30x6_)
InstrFetchProhibited	Instruction Access Fault	not in IRAM/RAM range	$PC(0x4_),$
		C	MEPC(0x30x6_)
•	Memory Protection Fault	write to IRAM or execute from	$MEPC(0x30x6_)$
		DRAM	
LoadProhibited	Load Access Fault	Read from NULL/invalid	EXCVADDR, MT-
		pointer	VAL
StoreProhibited	Store Access Fault	Save to NULL/invalid pointer	EXCVADDR, MT-
			VAL
LoadStoreAlignment	Load/Store Address Mis-	IRAM use as DRAM, but not 4-	EXCVADDR, MT-
	aligned	byte aligned	VAL
IntegerDivideByZero	•	calculate n/0	$PC(0x4_),$
			$MEPC(0x3\0x6_)$
LoadStoreError	•	write to read-only IROM	EXCVADDR, MT-
		DROM	VAL
Interrupt Watchdog	Interrupt Watchdog	Interrupt timeout/ disabled	
Timeout on CPU0/CPU1	Timeout on CPU0/CPU1	I I I I I I I I I I I I I I I I I I I	•
Cacha disabled but	Cache error	set ELAG IDAM but	$\mathbf{PC}(0rA)$
cached memory region	Cache error	not all data/functions in	$\frac{PC(0x4_{)}}{MEPC(0x3_{0}x6_{0})}$
accessed		IRAM/DRAM	WILL $C(0x3_{-}0x0_{-})$
Brownout	Brownout	Supply voltage lower than	Hardware
Diomiout	Diownout	threshold	Thirdware
rst:0x10	rst:0x10	no valid program in flash	Hardware
(RTCWDT_RTC_RESET)	(RTCWDT_RTC_RESET)		
rst:0x7	rst:0x7	Watchdog timeout	Flash or PSRAM
(TG0WDT_SYS_RST)	(TG0WDT_SYS_RST)		Pin / software
Corrupt Heap	Corrupt Heap	Heap corruption detected	Need further analy-
			sis
•	Stack protection fault	Stack overflow	Need further analy-
			sis
Stack smashing protect	Stack smashing protect	Stack overflow	Need further analy-
failure	failure		sis
Core 0 panic' ed Excep-	Stack canary watchpoint	Stack overflow	Need further analy-
tion was unhandled	triggered (task_name)		sis
UBSAN	UBSAN	Undefined behavior	Need further analy-
			sis

Table 3:	Xtensa ai	nd RISC-V	Exception	Mapping
14010 5.	1 Homba al		Enception	mapping

Core Dump Overview This section can be referred to in the Core Dump documentation. A brief summary is provided below:

- Core Dump is more comprehensive than backtrace, including stack dumps for each task.
- Core Dump can be printed to the terminal or saved to flash (type: data, subtype: coredump).
- After saving Core Dump to flash, it can be analyzed directly using the *idf.py coredump-info* command.
- The Core Dump serial data is in *BASE64* format as a binary string. Copy and save it as text, then use the *idf.py coredump-info -c </path/to/saved/base64/text>* command for analysis.
- esp monitor can parse Core Dump information and display it in a readable format.
- *idf.py coredump-debug* can "restore the scene," and GDB debugging will be opened to view variable values.

Xtensa Core Dump typically looks like the following:

```
Initiating core dump!
I (423) esp_core_dump_uart: Press Enter to print core dump to UART...
I (431) esp_core_dump_uart: Print core dump to uart...
Core dump started (further output muted)
Received 13 kB...
Core dump finished!
_____
The ROM ELF file won't load automatically since it was not found for the provided.
⇔chip type.
Crashed task handle: 0x3fc9a790, name: 'main', GDB name: 'process 1070180240'
exccause 0x1c (LoadProhibitedCause)
excvaddr 0x8
           0x40378347
epc1
           0x0
epc2
           0x0
epc3
epc4
           0x0
epc5
          0x0
epc6
           0x0
eps2
           0x0
eps3
           0x0
eps4
           0x0
eps5
           0x0
          0x0
eps6
          0x42018677
0x400556d5
                           0x42018677 <new_monkey_born+7>
рс
                           1074091733
lbeg
                           1074091749
          0x400556e5
lend
                            4294967292
lcount
           Oxffffffc
            0x1a
                            26
sar
ps
            0x60c20
                            396320
threadptr
br
            <unavailable>
            <unavailable>
scompare1
acclo
            <unavailable>
            <unavailable>
            <unavailable>
acchi
          <unavailable>
m0
          <unavailable>
m1
m2
           <unavailable>
mЗ
           <unavailable>
          <unavailable>
expstate
          <unavailable>
f64r_lo
f64r_hi
          <unavailable>
f64s
           <unavailable>
fcr
           <unavailable>
fsr
           <unavailable>
           0x8200a136
                            -2113887946
a0
                            1070179744
a1
           0x3fc9a5a0
                           0
            0x0
a2
           0x3fc9a7ec
                        1070180332
a3
            0x3c023088
                            1006776456
a4
                            1070179792
a5
            0x3fc9a5d0
                            1070179760
            0x3fc9a5b0
a6
a7
            0xc
                            12
            0x8200e2b8
a8
                            -2113871176
                           1070178896
            0x3fc9a250
a9
                            16
a10
            0x10
            Oxfffffff
                            -1
a11
a12
            0 \times 10
                             16
```

a13 0x3fc9a460 1070179424 a14 1070178912 0x3fc9a260 a15 0x1 1 #0 new_monkey_born (zoo=0x0) at /home/libo/test_github/idf_debug_method/main/idf_ →debug_method.c:21 #1 0x4200a136 in app_main () at /home/libo/test_github/idf_debug_method/main/idf_ →debug_method.c:57 #2 0x42019ade in main_task (args=<optimized out>) at /home/libo/esp/github_master/ ⇔components/freertos/app_startup.c:208 #3 0x4037a2ec in vPortTaskWrapper (pxCode=0x42019a38 <main_task>,__ →pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 Id Target Id Frame * 1 process 1070180240 new_monkey_born (zoo=0x0) at /home/libo/test_github/idf_ →debug_method/main/idf_debug_method.c:21 process 1070182124 vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /home/ 2 -->libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/xtensa/port. ⇔c:161 process 1070184008 0x40378326 in esp_cpu_wait_for_intr () at /home/libo/esp/ 3 →github_master/components/esp_hw_support/cpu.c:145 4 process 1070169660 0x400559e0 in ?? () 5 process 1070175732 0x400559e0 in ?? () 6 process 1070171288 0x400559e0 in ?? () #0 new_monkey_born (zoo=0x0) at /home/libo/test_github/idf_debug_method/main/idf_ →debug_method.c:21 #1 0x4200a136 in app_main () at /home/libo/test_github/idf_debug_method/main/idf_ →debug_method.c:57 #2 0x42019ade in main_task (args=<optimized out>) at /home/libo/esp/github_master/ →components/freertos/app_startup.c:208 #3 0x4037a2ec in vPortTaskWrapper (pxCode=0x42019a38 <main_task>,_ →pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 #0 vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /home/libo/esp/github_ →master/components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:161 #1 0x4000000 in ?? () ======================== THREAD 3 (TCB: 0x3fc9b648, name: 'IDLE1')_ ⇔====================== #0 0x40378326 in esp_cpu_wait_for_intr () at /home/libo/esp/github_master/ →components/esp_hw_support/cpu.c:145 #1 0x42002be5 in esp_vApplicationIdleHook () at /home/libo/esp/github_master/ ⇔components/esp_system/freertos_hooks.c:59 #2 0x4037b038 in prvIdleTask (pvParameters=0x0) at /home/libo/esp/github_master/ →components/freertos/FreeRTOS-Kernel/tasks.c:4170 #3 0x4037a2ec in vPortTaskWrapper (pxCode=0x4037b02c <prvIdleTask>,___ →pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 #0 0x400559e0 in ?? () #1 0x4037a692 in vPortClearInterruptMaskFromISR (prev_level=<optimized out>) at / -home/libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/xtensa/ →include/freertos/portmacro.h:582 (continues on next page)

#2 vPortExitCritical (mux=<optimized out>) at /home/libo/esp/github_master/ →components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:532 #3 0x4037c0c5 in xTaskGenericNotifyWait (uxIndexToWait=0, ulBitsToClearOnEntry= →<optimized out>, ulBitsToClearOnExit=4294967295, pulNotificationValue=0x3fc97ca0, → xTicksToWait=4294967295) at /home/libo/esp/github_master/components/freertos/ →FreeRTOS-Kernel/tasks.c:5644 #4 0x40378104 in ipc_task (arg=0x0) at /home/libo/esp/github_master/components/ →esp_system/esp_ipc.c:58 #5 0x4037a2ec in vPortTaskWrapper (pxCode=0x403780d4 <ipc_task>,_ →pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 _____ #0 0x400559e0 in ?? () #1 0x4037a692 in vPortClearInterruptMaskFromISR (prev_level=<optimized out>) at / →home/libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/xtensa/ \rightarrow include/freertos/portmacro.h:582 #2 vPortExitCritical (mux=<optimized out>) at /home/libo/esp/github_master/ →components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:532 #3 0x4037bf89 in ulTaskGenericNotifyTake (uxIndexToWait=0, xClearCountOnExit=1,_ -xTicksToWait=<optimized out>) at /home/libo/esp/github_master/components/ ⇔freertos/FreeRTOS-Kernel/tasks.c:5565 #4 0x42006143 in timer_task (arg=0x0) at /home/libo/esp/github_master/components/ →esp_timer/src/esp_timer.c:477 #5 0x4037a2ec in vPortTaskWrapper (pxCode=0x42006134 <timer_task>,__ →pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 #0 0x400559e0 in ?? () #1 0x4037a692 in vPortClearInterruptMaskFromISR (prev_level=<optimized out>) at / -home/libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/xtensa/ →include/freertos/portmacro.h:582 #2 vPortExitCritical (mux=<optimized out>) at /home/libo/esp/github_master/ #3 0x4037c0c5 in xTaskGenericNotifyWait (uxIndexToWait=0, ulBitsToClearOnEntry= →<optimized out>, ulBitsToClearOnExit=4294967295, pulNotificationValue=0x3fc98300, \hookrightarrow xTicksToWait=4294967295) at /home/libo/esp/github_master/components/freertos/ →FreeRTOS-Kernel/tasks.c:5644 #4 0x40378104 in ipc_task (arg=0x1) at /home/libo/esp/github_master/components/ →esp_system/esp_ipc.c:58 #5 0x4037a2ec in vPortTaskWrapper (pxCode=0x403780d4 <ipc_task>,_ →pvParameters=0x1) at /home/libo/esp/github_master/components/freertos/FreeRTOS-→Kernel/portable/xtensa/port.c:162 Name Address Size Attrs .rtc.force_fast 0x600fe010 0x0 RW .rtc_noinit 0x5000000 0x0 RW .rtc.force_slow 0x5000000 0x0 RW .iram0.vectors 0x40374000 0x403 R XA .iram0.text 0x40374404 0xd56f R XA .dram0.data 0x3fc91a00 0x3e84 RW A .flash.text 0x42000020 0x1a41b R XA .flash.appdesc 0x3c020020 0x100 R A .flash.rodata 0x3c020120 0xae5c RW A .flash.rodata_noload 0x3c02af7c 0x0 RW .ext_ram.bss 0x3c030000 0x0 RW .iram0.data 0x40381a00 0x0 RW

.iram0.bss 0x40381a00 0x0 RW						
. dramo. neap_scare 0x5res	6a20 0X0	01 E 4	DM			
.Coredump.tasks.data 0x5	109a/90 0	UXIJ4	RW			
.coredump.tasks.data 0x3	fc9a4e0 (0x2a0	RW			
.coredump.tasks.data 0x3	fc9aeec (0x154	RW			
.coredump.tasks.data 0x3	fc9ace0 (0x200	RW			
.coredump.tasks.data 0x3	fc9b648 (0x154	RW			
.coredump.tasks.data 0x3	fc9b3c0 (0x280	RW			
.coredump.tasks.data 0x3	fc97e3c (0x154	RW			
.coredump.tasks.data 0x3	fc97b90 (0x2a0	RW			
.coredump.tasks.data 0x3	fc995f4 (0x154	RW			
.coredump.tasks.data 0x3	fc99350 (0x290	RW			
.coredump.tasks.data 0x3	fc98498 (0x154	RW			
.coredump.tasks.data 0x3	fc981f0 (0x2a0	RW			
====== ES	P32 CORE	DUMP	END ====================================			
Done!						
Coredump checksum='7b594	5fe'					
I (1690) esp_core_dump_u	art: Core	e dump	p has been written to uar	t.		

Xtensa Core Dump typically looks like the following:

<pre>Initiating core dump! I (696) esp_core_dump_uart: Press Enter to print core dump to UART I (703) esp_core_dump_uart: Print core dump to uart Core dump started (further output muted) Received 3 kB Core dump finished! ====================================</pre>						
Crashed task ha	andle: 0x3	3fc91594 ,	, name: 'main', GDB name: 'process 1070142868'			
	==== CURRE	ENT THRE	AD REGISTERS ====================================			
ra	0x4200940)4	0x42009404 <app_main+56></app_main+56>			
sp	0x3fc9151	0	0x3fc91510			
gp	0x3fc8b40	00	0x3fc8b400 <c.29+52></c.29+52>			
tp	0x3fc8836	58	0x3fc88368			
t0	0x4005890)e	1074104590			
t1	0x3fc9110	õc	1070141804			
t2	0x0	0				
fp	0x3c02200	00	0x3c022000			
s1	0x3fc91f9	98	1070145432			
a0	0x0	0				
a1	0x3fc9114	18	1070141768			
a2	0x0	0				
a3	0x1	1				
a4	0x0	0				
a5	0x0	0				
аб	0x6002300	00	1610756096			
a7	0xa	10				
s2	0x0	0				
s3	0x0	0				
s4	0x0	0				
s5	0x0	0				
s6	0x0	0				
s7	0x0	0				
s8	0x0	0				
s9	0x0	0				
s10	0x0	0				

s11	0x0 0
t3	0x0 0
t4	0x0 0
t5	0x0 0
t6	0x0 0
pc	0x4200939e 0x4200939e <new_monkey_born+4></new_monkey_born+4>
#0 new_monkey_ →main/idf_debu	<pre>===== CURRENT THREAD STACK ====================================</pre>
#1 $0x42009404$ \leftrightarrow debug_method	<pre>in app_main () at /nome/libo/test_github/idf_debug_method/main/idfc:55 in main task (args=correr reading variable; value has heep</pre>
↔c:208	<pre>:>) at /home/libo/esp/github_master/components/freertos/app_startup.</pre>
#3 0x403845bc → <optimized ou<br="">→Kernel/portal</optimized>	<pre>in vPortTaskWrapper (pxCode=<optimized out="">, pvParameters= ut>) at /home/libo/esp/github_master/components/freertos/FreeRTOS- ole/riscv/port.c:234</optimized></pre>
Id Target Id	Frame
* 1 process →github/idf_de	1070142868 new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_ ebug_method/main/idf_debug_method.c:21
2 process 10 →pvParameters=)70144752 vPortTaskWrapper (pxCode=0x40384f20 <prvidletask>,_ =0x0) at /home/libo/esp/github master/components/freertos/FreeRTOS-</prvidletask>
↔Kernel/portal	ple/riscv/port.c:230
→level=1) at , →portable/rise	<pre>/home/libo/esp/github_master/components/freertos/FreeRTOS-Kernel/ cv/port.c:417</pre>
=====================================	<pre>===== THREAD 1 (TCB: 0x3fc91594, name: 'main') ====================================</pre>
#1 0x42009404 →debug_method	<pre>in app_main () at /home/libo/test_github/idf_debug_method/main/idf_ .c:55</pre>
#2 $0x42017976$ \rightarrow optimized out \rightarrow c:208	<pre>in main_task (args=<error been_<br="" has="" reading="" value="" variable:="">:>) at /home/libo/esp/github_master/components/freertos/app_startup.</error></pre>
#3 0x403845bc → <optimized ou<br="">→Kernel/portal</optimized>	<pre>in vPortTaskWrapper (pxCode=<optimized out="">, pvParameters= it>) at /home/libo/esp/github_master/components/freertos/FreeRTOS- ole/riscv/port.c:234</optimized></pre>
=====================================	===== THREAD 2 (TCB: 0x3fc91cf0, name: 'IDLE') ====================================
↔c:230 #1 0x00000000 Backtrace stops	in ?? () ped: frame did not save the PC
==================	THREAD 3 (TCB: 0x3fc903f8, name: 'esp timer')
= = = = = = = = = = = = = = = = = = =	in vPortClearInterruptMaskFromISR (prev int level=1) at /home/libo/
⇒esp/github_ma #1 0x403847f8 ⇒freertos/Free	aster/components/freertos/FreeRTOS-Kernel/portable/riscv/port.c:417 in vPortExitCritical () at /home/libo/esp/github_master/components/ eRTOS-Kernel/portable/riscv/port.c:517
#2 0x40385cfa →xClearCountOn	<pre>in ulTaskGenericNotifyTake (uxIndexToWait=uxIndexToWait@entry=0,_ nExit=xClearCountOnExit@entry=1,_</pre>
⇔xTicksToWait= ⇔components/fi	<pre>=xTicksToWait@entry=4294967295) at /home/libo/esp/github_master/ ceertos/FreeRTOS-Kernel/tasks.c:5565</pre>
#3 0x420046b6 ↔optimized out	in timer_task (arg= <error been_<br="" has="" reading="" value="" variable:="">>) at /home/libo/esp/github_master/components/esp_timer/src/esp_</error>
∟ ⇔ timer.c:477	(continues on next page)

```
#4 0x403845bc in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
→Kernel/portable/riscv/port.c:234
Name Address Size Attrs
.rtc.force_fast 0x50000010 0x0 RW
.rtc_noinit 0x50000010 0x0 RW
.rtc.force_slow 0x50000010 0x0 RW
.iram0.text 0x40380000 0xab8a R XA
.dram0.data 0x3fc8ac00 0x21c0 RW A
.flash.text 0x42000020 0x18504 R XA
.flash.appdesc 0x3c020020 0x100 R A
.flash.rodata 0x3c020120 0x9218 RW A
.eh_frame 0x3c029338 0xe0 R A
.flash.rodata_noload 0x3c029418 0x0 RW
.iram0.data 0x4038ac00 0x0 RW
.iram0.bss 0x4038ac00 0x0 RW
.dram0.heap_start 0x3fc8e490 0x0 RW
.coredump.tasks.data 0x3fc91594 0x154 RW
.coredump.tasks.data 0x3fc91470 0x110 RW
.coredump.tasks.data 0x3fc91cf0 0x154 RW
.coredump.tasks.data 0x3fc91c40 0xa0 RW
.coredump.tasks.data 0x3fc903f8 0x154 RW
.coredump.tasks.data 0x3fc90310 0xe0 RW
_____
Done!
Coredump checksum='704e2165'
I (1064) esp_core_dump_uart: Core dump has been written to uart.
```

Certainly, the Core Dump contains various detailed information, some of which is already included in the backtrace, such as the current Thread/Task registers and current Thread information. The additional need for Core Dump information often arises because it prints the state of all tasks running at the time of the current exception. This can be helpful in resolving issues like watchdog problems associated with tasks.

Locating Memory Issues (Memory Stompings, Memory Leaks, etc.)

Common memory issues include the following:

- Heap memory leak
- Heap memory stomping
- · Stack overflow
- Stack memory stomping
- Pointer used after being freed
- Pointer used before being initialized
- Double free

This section will explain the general debugging methods for the above memory issues one by one.

Heap Memory Leak Heap memory leaks often manifest as a program allocating a block of heap memory, but not correctly freeing it after use, leading to a memory leak. This can cause the memory consumed by the program to gradually increase during runtime, and may eventually exhaust the system' s available memory.

This part can refer to the Heap Memory Debugging Document, the key points are summarized as follows:

• You can use *heap_caps_get_per_task_info* to get the memory allocation status of all tasks

- You can use *heap_caps_get_free_size* to compare the remaining memory situation and roughly determine the leak area
- Enable CONFIG_HEAP_TRACING_STANDALONE or CONFIG_HEAP_TRACING_TOHOST
- STANDALONE mode requires buffer allocation, directly record, calculate, and print results on ESP, but RISC-V architecture cannot locate code lines
- TOHOST requires UART/JTAG to use app_trace capture, analyze on the host, no extra buffer is needed, code lines can be located
- heap trace init standalone initializes the buffer, heap trace start(HEAP TRACE LEAKS) starts recording
- *heap_trace_stop()* stops recording, use *heap_trace_dump()* to print analysis results

The typical log after using the above heap memory debugging method is as follows:

```
1. Xtensa
```

```
===== Heap Trace: 2 records (100 capacity) ======
36 bytes (@ 0x3fc9c524, Internal) allocated CPU 0 ccount 0x02f204e0 caller_
→0x42008cfd:0x42008d73
0x42008cfd: zoo_create at /home/libo/test_github/idf_debug_method/main/idf_debug_
→method.c:68
0x42008d73: mem_leak_task at /home/libo/test_github/idf_debug_method/main/idf_
→debug_method.c:96 (discriminator 3)
    24 bytes (@ 0x3fc9c54c, Internal) allocated CPU 0 ccount 0x02f20c00 caller_
→0x42008cfd:0x42008d73
0x42008cfd: zoo_create at /home/libo/test_github/idf_debug_method/main/idf_debug_
→method.c:68
0x42008d73: mem_leak_task at /home/libo/test_github/idf_debug_method/main/idf_
→debug_method.c:96 (discriminator 3)
===== Heap Trace Summary ======
Mode: Heap Trace Leaks
60 bytes 'leaked' in trace (2 allocations)
records: 2 (100 capacity, 3 high water mark)
total allocations: 3
total frees: 1
   _____
```

2. RISC-V

Heap Memory Stomping Heap memory stompings often occur when writing to or reading from heap memory, the program accesses an area beyond the memory range allocated to it. This can lead to undefined behavior and corrupt the program' s memory structure. The corresponding log for this error is often:

assert failed: remove_free_block tlsf.c:331 (next && "next_free field can not be_ →null")

This part can refer to the Heap Memory Debugging Document, the key points are summarized as follows:

1. Locate Who and Where

- Enable memory debugging, raise the heap memory debugging level CON-FIG_HEAP_CORRUPTION_DETECTION to *light impact* or *comprehensive*:
 - Basic (default): Use heap properties to detect if it has been contaminated
 - Light impact : Add special bytes 0xABBA1234 0xBAAD5678 before and after the allocated memory
 - Comprehensive : Add uninitialized-access and use-after-free bugs check on the basis of light impact. When memory is allocated, all memory is initialized to 0xce, and all spaces are assigned to 0xfe after memory is released
- After enabling memory debugging, wait for crash or actively call to check memory integrity *heap_caps_check_integrity_all* to trigger crash before and after the suspected memory stomping position. If the stomping address has been located, you can directly use *heap_caps_check_integrity_addr*.
 - Step on the tail, the current memory block operation is out of bounds CORRUPT HEAP: Bad tail at 0x3fc9ad5a. Expected 0xbaad5678 got 0x02020202
 - Step on the head, the previous memory block is out of bounds *CORRUPT HEAP: Bad head at 0x3fc9a94c*. *Expected 0xabba1234 got 0x00000000*
- Two methods can confirm the neighbors before and after the memory block
 - Use heap trace, call *heap_trace_start(HEAP_TRACE_ALL)* to collect information
 - Use *heap_caps_dump_all* to print the collected information (need to print before and after memory allocation)

Note: The specific method of confirming the memory block status described above can refer to Heap Memory Tracking.

- 2. Locate When
 - You can set the CPU breakpoint in the code through *esp_cpu_set_watchpoint(0, (void* *)0x3fc9a94c, 4, *ESP_CPU_WATCHPOINT_STORE);*. If you don't know which kernel, you need to call both kernels
 - The CPU will trigger a breakpoint when data is written to this address, and the code line can be located through PC, refer to the log as follows:

```
Guru Meditation Error: Core 0 panic'ed (Unhandled debug exception).
Debug exception reason: Watchpoint 0 triggered
Core 0 register dump:
PC
       : 0x400570e8 PS : 0x00060c36 A0
                                                : 0x82008d43 A1
→: 0x3fc99f10
0x400570e8: memset in ROM
A2
       : 0x3fc9b3ac A3
                            : 0x0000000 A4
                                                  : 0x00003e8
                                                               Α.5
↔: 0x3fc9b75c
A6
      : 0x0000000 A7
                            : 0x000003e A8
                                                  : 0x8200333d A9
⇔: 0x3fc99ee0
A10
      : 0x00000400 A11
                            : 0x00060c20 A12
                                                  : 0x0000000 A13
→: 0x00060c23
      : 0xb33fffff A15
A14
                            : 0xb33fffff SAR
                                                  : 0x0000004 🗆
→EXCCAUSE: 0x0000001
EXCVADDR: 0x0000000 LBEG
                            : 0x400570e8 LEND
                                                  : 0x400570f3 LCOUNT _
→: 0x0000002
Backtrace: 0x400570e5:0x3fc99f10 0x42008d40:0x3fc99f20_
→0x4201874b:0x3fc99f50 0x4037a80d:0x3fc99f80
0x400570e5: memset in ROM
0x42008d40: app_main at /home/libo/test_github/idf_debug_method/main/idf_
→debug_method.c:169 (discriminator 3)
0x4201874b: main_task at /home/libo/esp/github_master/components/freertos/
→app_startup.c:208 (discriminator 13)
0x4037a80d: vPortTaskWrapper at /home/libo/esp/github_master/components/
⇔freertos/FreeRTOS-Kernel/portable/xtensa/port.c:162
```

Stack Overflow The manifestation of stack overflow often occurs when using stack memory during function calls. If recursion or too many local variables are involved, the stack size may exceed the system's allowed limit, resulting in a stack overflow. The ESP-IDF provides the following stack overflow detection mechanisms:

1. ESP-IDF FreeRTOS enables stack overflow detection by default. If a stack overflow is detected, it triggers an assertion, printing the corresponding stack overflow information. A typical log looks like this:

```
***ERROR*** A stack overflow in task test_task has been detected.
```

For more details, refer to the Stack Overflow section.

- 2. ESP-IDF supports enabling the End of Stack Watchpoint, which triggers a breakpoint before FreeRTOS stack overflow assertion.
- 3. The RISC-V platform supports enabling hardware stack overflow detection (*Stack protection fault*). For details, see Hardware Stack Protection.

Stack Memory Stomping The manifestation of stack memory stomping is similar to heap memory stomping but occurs when the program uses stack memory. Writing or reading data beyond the allocated stack memory range may lead to program errors. Here are some key points to note:

- 1. It may lead to task stack overflow, generally detectable through FreeRTOS stack overflow mechanisms.
- 2. It may result in the overwriting of local variable values, causing unexpected program behavior.
- 3. It may cause the modification of local pointer variables, accessing illegal instructions/data addresses, leading to program crashes.
- 4. It may result in the overwriting of the function return address, causing the program to jump to an incorrect address and crash.

A simple example of error code is as follows:

```
int vulnerableFunction() {
    int localArray[5]; // Array allocated on the stack
    // Writing beyond the bounds of the array
    for (int i = 0; i <= 5; ++i) {
        localArray[i] = i;
    }
    return localArray[0];
}
void app_main() {
    printf("Before vulnerable function.\n");
    vulnerableFunction(); // Call the function that causes stack memory corruption
    printf("After vulnerable function.\n");
}</pre>
```

It's worth mentioning that ESP-IDF performs partial compile-time checks for such errors and issues warnings (though the compilation still passes). The compile-time warning log looks like this:

Pointers Used After Release The manifestation of using pointers after their release often occurs when a program releases a block of memory but continues to use a pointer that points to that memory. This may result in accessing invalid memory, leading to crashes or undefined behavior.

This issue can cause various errors that are challenging to trace through actual errors. Therefore, it is crucial to pay special attention to pointer usage during the development process. A simple example of erroneous code is as follows:

```
void app_main(void)
{
    int *number = (int *)malloc(sizeof(int)); // Allocate memory for an integer
    if (number == NULL) {
        // Handle memory allocation failure
        printf("Memory allocation failed.\n");
    }
    *number = 42; // Assign a value to the allocated memory
    printf("Value before freeing: %d\n", *number);
    free(number); // Free the allocated memory
    // Attempt to use the pointer after freeing
    // This will result in undefined behavior
    printf("Value after freeing: %d\n", *number);
}
```

It's worth noting that ESP-IDF can detect some of these errors during compilation and issue warnings (though the compilation may still succeed). Compilation-time warning logs may look like this:

Pointers Used Before Initialization The manifestation of using pointers before their initialization often occurs when a program attempts to use an uninitialized pointer, resulting in access to unknown memory regions and causing unstable behavior.

This issue can cause various errors that are challenging to trace through actual errors. Therefore, it is crucial to pay special attention to pointer initialization during the development process. A simple example of erroneous code is as follows:

```
void app_main(void)
{
    int *number; // Pointer declared but not initialized
    // Attempt to dereference the uninitialized pointer
    // This will result in undefined behavior
    printf("Value: %d\n", *number);
}
```

It's worth noting that ESP-IDF often detects some of these errors during compilation and issues error messages. Compilation-time error logs may look like this:

Double Free The manifestation of double-free often occurs when a program releases memory that has already been freed. This can lead to memory pool corruption, resulting in program crashes or other severe issues. An example of erroneous code is as follows:

```
void app_main(void)
{
    // Allocate a block of memory
    int *data = (int *)malloc(sizeof(int));
    // Check if memory allocation is successful
    if (data != NULL) {
        // Assign a value to the allocated memory
        *data = 42;
        // First free
        free(data); // Line 26
        // Second free (double-free)
        free(data); // Line 29, this is incorrect and may lead to undefined.
        --behavior
        }
}
```

It's worth noting that ESP-IDF often detects some of these errors during compilation and issues warning messages. Compilation-time warning logs may look like this:

Runtime error logs may look like this:

```
I (285) main_task: Calling app_main()

assert failed: tlsf_free tlsf.c:1119 (!block_is_free(block) && "block already_

→marked as free")

Core 0 register dump:

Stack dump detected

MEPC : 0x403805d8 RA : 0x403838e8 SP : 0x3fc8f330 GP :_

→0x3fc8ae00
```

```
0x403805d8: panic_abort at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/
→components/esp_system/panic.c:452
0x403838e8: __ubsan_include at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/
→components/esp_system/ubsan.c:313
       : 0x3fc87110 T0
                          : 0x37363534 T1
                                               : 0x7271706f T2
ТΡ
                                                                    ÷. .
↔0x33323130
S0/FP : 0x0000069 S1
                          : 0x0000001 A0
                                               : 0x3fc8f36c A1
                                                                    <u>۔</u>
⇔0x3fc8acd1
A2
       : 0x0000001 A3
                           : 0x0000029 A4
                                               : 0x0000001 A5
                                                                    <u>۔</u>
→0x3fc8c000
A6
      : 0x7a797877 A7
                           : 0x76757473 S2
                                               : 0x0000009 S3
                                                                    ↔0x3fc8f49e
      : 0x3fc8acd0 S5
                           : 0x0000000 S6
                                               : 0x0000000 S7
S4
                                                                    : __
→0x00000000
     : 0x0000000 S9
                          : 0x0000000 S10
S8
                                               : 0x0000000 S11
                                                                    : ....
⇔0x0000000
ΤЗ
     : 0x6e6d6c6b T4
                          : 0x6a696867 T5
                                               : 0x66656463 T6
                                                                    →0x62613938
MSTATUS : 0x00001881 MTVEC : 0x40380001 MCAUSE : 0x00000007 MTVAL
                                                                    ÷. .
↔0x0000000
0x40380001: _vector_table at ??:?
MHARTID : 0x0000000
Backtrace:
panic_abort (details=details@entry=0x3fc8f36c "assert failed: tlsf_free tlsf.
→zhengzhong/github/esp-idf/rel5.1/esp-idf/components/esp_system/panic.c:452
452
           *((volatile int *) 0) = 0; // NOLINT(clang-analyzer-core.
\hookrightarrowNullDereference) should be an invalid operation on targets
#0 panic_abort (details=details@entry=0x3fc8f36c "assert failed: tlsf_free tlsf.
\rightarrowc:1119 (!block_is_free(block) && \"block already marked as free\")") at /home/
-->zhengzhong/github/esp-idf/rel5.1/esp-idf/components/esp_system/panic.c:452
#1 0x403838e8 in esp_system_abort (details=details@entry=0x3fc8f36c "assert_
→failed: tlsf_free tlsf.c:1119 (!block_is_free(block) && \"block already marked_
→as free\")") at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/components/esp_
→system/port/esp_system_chip.c:84
#2 0x403890e8 in __assert_func (file=file@entry=0x3c0212f3 "",_
→line=line@entry=1119, func=<optimized out>, func@entry=0x3c021984 <__func__.6> "
→idf/components/newlib/assert.c:81
#3 0x40387e5e in tlsf_free (tlsf=0x3fc8c574, ptr=ptr@entry=0x3fc8ff20) at /home/
→zhengzhong/github/esp-idf/rel5.1/esp-idf/components/heap/tlsf/tlsf.c:1119
#4 0x40387a8e in multi_heap_free_impl (heap=0x3fc8c560, p=p@entry=0x3fc8ff20) at /

-home/zhengzhong/github/esp-idf/rel5.1/esp-idf/components/heap/multi_heap.c:231

#5 0x40380b98 in heap_caps_free (ptr=ptr@entry=0x3fc8ff20) at /home/zhengzhong/
→github/esp-idf/rel5.1/esp-idf/components/heap/heap_caps.c:388
#6 0x4038910e in free (ptr=ptr@entry=0x3fc8ff20) at /home/zhengzhong/github/esp-
→idf/rel5.1/esp-idf/components/newlib/heap.c:39
#7 0x4200712a in app_main () at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/
wamples/get-started/hello_world/main/hello_world_main.c:29
#8 0x4201498a in main_task (args=<error reading variable: value has been_
-optimized out>) at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/components/

→freertos/app_startup.c:208

#9 0x40385a2c in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
--<optimized out>) at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/components/
(continues on next page)
```

```
ELF file SHA256: 1df25094bc6834da
```

You can see from the logs that there is a *block already marked as free* warning and an error code location hint at *app_main () at /home/zhengzhong/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/hello_world_main.c:29.* It indicates that a double-free situation occurred at line 29, and the second call to *free* needs to be removed.

Using SystemView for System Analysis and Optimization

This section can be directly referred to the Application Layer Trace Library chapter in the ESP-IDF Programming Guide.

2.2.3 Performance Optimization

The section introduces common performance optimization methods for ESP chips.

ESP Chip Startup Time Optimization

Introduction The startup time of ESP chips refers to the time it takes for the ESP chip to power on and execute the *app_main* function. The unoptimized startup process usually takes a long time (generally around 300 milliseconds), making it unable to meet the requirements of applications with high real-time requirements (such as turning on/off lights at any time). At the same time, it also accompanies high power consumption, resulting in high average power consumption in Deep-sleep low-power mode. To avoid these problems, fine optimization of the startup process is necessary.

Hardware and Software Environment

- Hardware: ESP32-S3 and ESP32-C6
- Software: ESP-IDF v5.2.1

Optional Optimization Items Configuring the following options through *menuconfig* can significantly reduce startup time:

- 1. Disable Bootloader Log Printing
 - CONFIG_BOOTLOADER_LOG_LEVEL_NONE=y
 - CONFIG_BOOTLOADER_LOG_LEVEL=0
- 2. Skip Image Validation
 - CONFIG_BOOTLOADER_SKIP_VALIDATE_IN_DEEP_SLEEP=y
 - CONFIG_BOOTLOADER_SKIP_VALIDATE_ON_POWER_ON=y
 - CONFIG_BOOTLOADER_SKIP_VALIDATE_ALWAYS=y
- 3. Disable Boot ROM Log Printing
 - CONFIG_BOOT_ROM_LOG_ALWAYS_OFF=y

In addition to this configuration, you also need to use the *espefuse.py* command in the terminal to configure the eFuse values related to controlling ROM log output (see the Boot Log Printing Control section of the *Technical Reference Manual <https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf>`__ for details; note that writing eFuse operations is irreversible). The specific command is as follows:*

espefuse.py burn_efuse UART_PRINT_CONTROL 3
espefuse.py burn_efuse DIS_USB_SERIAL_JTAG_ROM_PRINT 1

After running the above two commands, you can use the *espefuse.py summary* command to check whether the writing is successful. If you see the following information, it means the configuration is successful:

- 4. Modify SPI Flash Mode and Frequency
 - CONFIG_ESPTOOLPY_FLASHMODE_QIO=y
 - CONFIG_ESPTOOLPY_FLASHFREQ_80M=y
- 5. Disable Boot Calibration
 - CONFIG_ESP_PHY_CALIBRATION_AND_DATA_STORAGE=y
 - CONFIG_ESP_PHY_CALIBRATION_MODE=1
- 6. Modify FreeRTOS Configuration
 - CONFIG_FREERTOS_UNICORE=y
 - CONFIG_FREERTOS_HZ=1000
- 7. Disable Log Printing
 - CONFIG_LOG_DEFAULT_LEVEL_NONE=y
 - CONFIG_LOG_DEFAULT_LEVEL=0

If further optimization of the time to connect to the AP after power-on is required on the basis of optimizing the startup time, the following operations can be performed synchronously:

1. Enable LWIP_DHCP_RESTORE_LAST_IP Configuration Option

• CONFIG_LWIP_DHCP_RESTORE_LAST_IP=y

Startup Time and Average Power Consumption During Startup Results Statistics Startup time and average power consumption statistics of ESP32-S3 at different main frequencies:

CPU Frequency	80 M	160 M	240 M
Startup Time Before Optimization (ms)	318.8	318.7	318.7
Startup Power Consumption Before Opti-	44.1	51.4	57.3
mization (mA)			
Startup Time After Optimization (ms)	26.87	26.875	26.965
Startup Power Consumption After Opti-	29.69	29.77	29.74
mization (mA)			

Startup time and average power consumption statistics of ESP32-C6 at different main frequencies:

CPU Frequency	80 M	120 M	160 M
Startup Time Before Optimization (ms)	328.0	327.9	327.8
Startup Power Consumption Before Opti-	34.1	35.8	37.6
mization (mA)			
Startup Time After Optimization (ms)	33.31	33.315	33.315
Startup Power Consumption After Opti-	28.23	28.19	28.16
mization (mA)			

ESP Low Power Parameter Configuration

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Introduction In Light Sleep mode, the RF module is turned off, and most modules, except for RTC-related ones, are powered down or undergo dynamic frequency adjustment to achieve power reduction. Some menuconfig can be configured as needed, and this article will introduce these configuration items. The chip' s base current at this time can refer to the power consumption on the datasheet, such as ESP32-C3 is 130 μ A, ESP32-S3 is 240 μ A, ESP32-C6 is 35 μ A (the actual measurement is 45 μ A, the subsequent datasheet will correct this item).

1. Common Power Optimization Configuration Options

1.1. Dynamic Frequency Adjustment The higher the frequency at which the CPU works, the greater the power consumption. Through DFS (dynamic frequency scaling), the system can automatically switch the working frequency to balance power consumption and performance. To enable this feature, you need to enable the two items shown in Figure 2-1, so the system will switch to the maximum frequency when working, and switch to the lowest frequency when the system is idle. The maximum frequency is the default main frequency, such as ESP32-C3 is 160 MHz, and the lowest frequency is the frequency of the external main crystal oscillator, generally 40 MHz. The maximum and minimum frequencies can be set by calling esp_pm_configure.

Note: In the scenario of automatic frequency tuning for Wi-Fi applications, it is necessary to ensure that the minimum frequency is not lower than 40 MHz. When the frequency is below 40 MHz, the processing speed of the CPU will be significantly reduced, and the optimization effect in terms of low power consumption will not be obvious either.

Figure 2-1 Dynamic Frequency Adjustment Configuration Items

- Configuration item name: PM_ENABLE
- Configuration item path: (Top) -> Component config -> Power Management -> Support for power management
- Configuration item name: PM_DFS_INIT_AUTO
- Configuration item path: (Top) -> Component config -> Power Management -> Support for power management -> Enable dynamic frequency scaling (DFS) at startup

1.2 Automatic light sleep If you need to further reduce power consumption after enabling dynamic frequency adjustment, as shown in Figure 2-1, you need to enable automatic light sleep. Modem sleep mode plus automatic light sleep is what we often call power save mode.

Figure 2-2 Automatic light sleep configuration items

- Configuration item name: FREERTOS_HZ
- Configuration item path: (Top) -> Component config -> FreeRTOS -> Tick rate (Hz)
- Configuration item name: FREERTOS_USE_TICKLESS_IDLE
- Configuration item path: (Top) -> Component config -> FreeRTOS -> Tickless idle support

This involves two configuration items, item A is the Tick frequency of freeRTOS, the default is 100, that is, each Tick requires 1000 / 100=10 ms. Item B is the minimum idle Tick number required to automatically enter light sleep, the default is 3, that is, when the system detects that the idle time is more than 3 Ticks, it will automatically enter light sleep. As shown in the figure above, when the idle time is more than 3*10=30 ms, the system will automatically enter light sleep. By increasing item A, the system can more sensitively detect idle time and enter sleep. For example, if the tick rate is configured as 1000, that is, if there is no task working within 3 ms, it will enter sleep, thereby achieving lower power consumption.

Note:

- 1. Automatic entry into light sleep can only be configured after enabling the configuration items in 2.1.
- 2. After enabling automatic frequency adjustment and automatic sleep, the communication of some peripherals will be affected. For specific details, please refer to the programming guide.

The most common question from customers is how to use LEDC and UART while maintaining power consumption. The solutions are: - For customers who want to use UART interrupts during automatic sleep, first ask them if they have any spare GPIO pins, and recommend GPIO wake-up as a priority. - Ask the customer if there is an external 32K crystal oscillator on the hardware. If there is, change the peripheral clock source to the external 32K crystal oscillator. - Use locks, which can disable dynamic frequency adjustment during communication. See attachment 2 for an example.

1.3 Isolate GPIO During system sleep, GPIO leakage will cause current loss and increase system power consumption. In esp-idf, GPIO leakage is eliminated during light sleep by floating (disabling internal pull-up and pull-down resistors) and isolating (disconnecting pin input and output) GPIO pins. The corresponding menuconfig configuration item is as follows.

Note: This configuration item can only be enabled after enabling automatic sleep in 2.2.

Figure 2-3 Isolate GPIO Configuration Item

- Configuration item name: PM_SLP_DISABLE_GPIO
- Configuration item path: (Top) -> Component config -> Power Management -> Disable all GPIO when chip at sleep

When this option is enabled, all GPIO pins will be disabled during system sleep, eliminating the impact of GPIO leakage on sleep power consumption, but also making it impossible for GPIO to input and output signals during sleep. However, in some applications, the application layer hopes to be able to use GPIO functions (input/output/internal pull-up and pull-down) normally during system sleep. Therefore, IDF provides a set of APIs for managing GPIO status during sleep. For related APIs, refer to Table 2-1.

Table 2-1 Sleep Process GPIO Management API

• Set the input and output status of GPIO in sleep state:

esp_err_t gpio_sleep_set_direction(gpio_num_t gpio_num, gpio_mode_t mode);

• Set the pull-up and pull-down status of GPIO in sleep state:

• Enable automatic GPIO status switching:

esp_err_t gpio_sleep_sel_en(gpio_num_t gpio_num);

• Disable automatic GPIO status switching:

esp_err_t gpio_sleep_sel_dis(gpio_num_t gpio_num);

If you want to maintain the status of the GPIO pin during sleep, such as controlling light output and controlling switch closure, you can use the gpio_hold_en(gpio_num_t gpio_num) and gpio_hold_dis(gpio_num_t gpio_num) APIs to hold the GPIO that needs to maintain the level before entering sleep, and hold_dis after waking up.

1.4 Flash Power Down ESP32 expands system storage resources through external flash and PSRAM. Flash has the characteristic of not losing data after power loss, while PSRAM cannot retain data after power loss. When PSRAM is not used in the system, the flash power-down function can be enabled to reduce the sleep current of the chip. The related menuconfig options are as follows:

Figure 2-4 Flash Power Down Configuration Item

- Configuration item name: ESP_SYSTEM_PD_FLASH (release/v4.3)
- Configuration item name: ESP_SLEEP_POWER_DOWN_FLASH (4.4 and later)
- Configuration item path: (Top) -> Component config -> ESP System Settings -> PD flash at light sleep when there is no SPIRAM (release/v4.3)
- Configuration path: (Top) -> Component config -> Hardware Settings -> Sleep Config -> Power down flash in light sleep when there is no SPIRAM (4.4 and later)

Note: In reality, the time required for flash power-off is difficult to predict. Even if the time required for complete flash power-off can be known, it is sometimes impossible to ensure the safety of flash power-off by setting a sufficiently long sleep time (for example, sudden asynchronous wake-up sources can make the actual sleep time uncontrollable). At this time, the hardware behavior of re-powering before the power-off is completed may cause the flash to fail to work normally. Therefore, the following adjustments can be made to the configuration items:

Figure 2-5 Flash Power-off Configuration Optimization

- Configuration name: ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND
- Configuration path: (Top) -> Component config -> Hardware Settings -> Sleep Config -> Pull-up Flash CS pin in light sleep

This method is better than powering off the flash, and it increases the base current consumption by tens of μA , but it takes into account both safety and power consumption. If a filter capacitor is added to the power supply circuit of the flash, it should be avoided as much as possible to power off the flash.

1.5 CPU Power Down For applications with higher power consumption requirements in light sleep mode, the CPU power-down function can be enabled during sleep to reduce the sleep current. For ESP32-C3, the sleep current is reduced by about 100 μ A, and for ESP32-S3, the sleep current is reduced by about 650 μ A. Currently, it is only available on these two chips, and ESP32 and ESP32-S2 cannot use it.

Powering down the CPU during sleep will cause the loss of CPU execution context information (dedicated, general, and status registers, etc.), causing the CPU to execute abnormally after waking up from sleep. Therefore, it is necessary to back up the context information to the retention memory before the CPU is powered down. After the system wakes up, the CPU executes the context information from the retention memory before execution. Backing up the CPU context information requires system memory resources. For ESP32-C3, it will consume 1.6 KB of DRAM space, and for ESP32-S3, it will consume 8.58 KB of DRAM space.

Figure 2-6 CPU Power Down Configuration

- Configuration name: PM_POWER_DOWN_CPU_IN_LIGHT_SLEEP
- Configuration path: (Top) -> Component config -> Power Management -> Power down CPU in light sleep

1.6 Peripheral Power Down (light sleep) By turning off this item, the base current can be reduced by about 100 μ A.

- Configuration name: PM_POWER_DOWN_PERIPHERAL_IN_LIGHT_SLEEP
- **Configuration path**: (Top) -> Component config -> Power Management -> Power down Digital Peripheral in light sleep (EXPERIMENTAL)

When the peripheral power domain is powered off during sleep, the IO_MUX and GPIO modules are both powered off, which means that the state of the chip pins will not be controlled by these modules. To maintain the state of IO during sleep, you need to call $gpio_hold_dis()$ and $gpio_hold_en()$ before and after configuring the GPIO state. This operation can ensure that the IO configuration is latched, preventing IO from floating during sleep.

2. Wi-Fi/Bluetooth Low Power Optimization Configuration Espressif chips are wireless MCUs, so most of the time, low power functions need to be used with Wi-Fi/Bluetooth functions. Here, some low power configurations and parameters related to Wi-Fi/Bluetooth are explained.

2.1 Bluetooth Low Power Optimization Items

2.1.1 BLE sleep Currently, there are few optimization configurations available for the Bluetooth end, but BLE modem sleep can be enabled.



• Configuration path: - (Top) -> Component config -> Bluetooth -> Bluetooth -> Bluetooth controller -> MODEM SLEEP Options - (Top) → Component config → Bluetooth → Controller Options(V5.2)

If you want to enter light sleep instead of modem sleep during keep-alive, you can refer to the method in 2.2.

It should be noted that in IDF versions 4.4 and later, Bluetooth already supports using the main crystal as the sleep clock. The cost of not using an external 32 KHz crystal is that the base current will rise during light sleep. Reference current: - ESP32-C3: 2.3 mA - ESP32-S3: 3.3 mA



2.1.2 Nimble – BLE only

- Configuration item name: **BT_NIMBLE_ENABLED**
- Configuration path: (Top) -> Component config -> Bluetooth -> Bluetooth -> Host

In the BLE only scenario, this item can be configured to reduce memory consumption.

2.2 Wi-Fi Low Power Optimization There are many configuration items involved in Wi-Fi optimization, and the following are more common:

2.2.1 Power Down Wireless Digital Circuit After enabling automatic light sleep, this option can be turned on to power down the wireless digital circuit module when the physical layer is turned off, which can save about 100 μ A of base current.

(Top) → Component config → PHY
[*] Store phy calibration data in NVS [] Use a partition to store PHY init data
(20) Max WiFi TX power (dBm) [] Power down MAC and baseband of Wi-Fi and Bluetooth when PHY is disabled (NEW)
[*] Enable USB when phy init
Configuration item name: ESP_PHY_MAC_BB_PD

• Configuration path: (Top) -> Component config -> PHY -> Power down MAC and baseband of Wi-Fi and Bluetooth when PHY is disabled

2.2.2 Change Minimum Wait Time and Maximum Keep-Alive Time When the ESP32 operates as both STA and AP, it needs to wait for a period of time before turning off the RF after receiving data once, this time is called the Minimum active time, the default value is 50 ms. On the basis of ensuring the data throughput each time, this waiting time can be appropriately reduced to lower power consumption.

When the ESP32 is in power save mode, it needs to periodically send a keep alive packet to the AP to tell the AP that it is still connected. The Maximum keep alive time is the time period for sending this keep alive packet, the default is 10 s. Power consumption can be reduced by increasing this parameter to reduce packet sending.



- Configuration item name: ESP_WIFI_SLP_DEFAULT_MIN_ACTIVE_TIME
- Configuration path: (Top) -> Component config -> Wi-Fi -> Wi-Fi SLP IRAM speed optimization -> Minimum active time
- Configuration item name: ESP_WIFI_SLP_DEFAULT_MAX_ACTIVE_TIME
- Configuration path: (Top) -> Component config -> Wi-Fi -> Wi-Fi SLP IRAM speed optimization -> Maximum keep alive time

Recommended configuration: Minimum active time = 15~20, Maximum keep alive time = 60.

2.2.3 Enable Power Management for Disconnection After enabling this configuration item, the Modem sleep state can still be maintained when a disconnection occurs.



- Configuration item name: ESP_WIFI_STA_DISCONNECTED_PM_ENABLE
- Configuration path: (Top) → Component config → Wi-Fi → Power Management for station at disconnected

2.2.4 Enable Beacon Lost Optimization (only for release/v4.4 and above) One of the basics of Wi-Fi keep alive is to be able to receive the beacon packet sent by the router on time. In actual use environment, various reasons can cause beacon loss. In versions prior to release/v4.4, after losing the beacon, the ESP32 will keep the RF on until the next beacon is received. Versions after 4.4 can avoid the impact of this behavior on power consumption by enabling beacon lost optimization.

[*] Wif	i sleep optimize when beacon lost
(10)	Beacon loss timeout (NEW)
(3)	Maximum number of consecutive lost beacons allowed (NEW)
(2)	Delta early time for RF PHY on (NEW)
(2)	Delta timeout time for RF PHY off (NEW)

- Configuration item name: ESP_WIFI_SLP_BEACON_LOST_OPT
- Configuration path: (Top) → Component config → Wi-Fi → Wifi sleep optimize when beacon lost

As shown in Figure 2-6, the optimization principle is that the RF receiver will not always be on when the beacon is lost. Instead, it will be turned on for a while and if no reception is received, it will go to sleep, then wake up to receive packets. If it still hasn' t received anything after repeating this process a few times, it will operate as usual, keeping the RF on until it receives a position. This can greatly improve the fault tolerance rate when the beacon is not received, thereby reducing power consumption.





After enabling this option, four additional configuration items will be introduced, mainly focusing on the first two configuration items:

- Option A: **Beacon loss timeout** indicates how long to wait without receiving a beacon before entering sleep mode.
- Option B: Maximum number of consecutive lost beacons allowed indicates how many times to repeat without receiving before continuously turning on.

Note that option B should not be set too high, generally set to 3.

2.2.5 Reduce TX power After enabling this option, the PHY TX power will be reduced after detecting a brownout reset, allowing the code to continue running.

(Top) → Component config → PHY
Espressif IoT Development Framework Configuration
[*] Store phy calibration data in NVS
[] Use a partition to store PHY init data
(10) Max WiFi TX power (dBm)
[*] Reduce PHY TX power when brownout reset
[] Enable USB when phy init
Calibration mode (Calibration partial)>

• Configuration item name: ESP_PHY_REDUCE_TX_POWER

• Configuration path: (Top) → Component config → PHY → Reduce PHY TX power when brownout reset

2.2.6 Optimize Sleep IRAM Speed Selecting this option will place the Wi-Fi library's TBTT process and receive beacon function in IRAM. If this option is enabled, the average current of the Wi-Fi powersave mode will be reduced.

(Top) → Component config → Wi-Fi
↑↑↑↑↑↑↑↑↑↑↑↑↑↑ Espressif IoT Development Framework Configuration
[*] WiFi RX IRAM speed optimization
[*] Enable WPA3-Personal
[*] Enable SAE-PK
[*] Enable WPA3 Personal(SAE) SoftAP
[*] Enable OWE STA
[*] WiFi SLP IRAM speed optimization

• Configuration item name: ESP_WIFI_SLP_IRAM_OPT

• Configuration path: (Top) → Component config → Wi-Fi → WiFi SLP IRAM speed optimization

2.2.7 Optimize IRAM Speed All three of the following items can be enabled.



- Configuration item name: ESP_WIFI_IRAM_OPT ESP_WIFI_EXTRA_IRAM_OPT ESP_WIFI_RX_IRAM_OPT
- Configuration Path: (Top) → Component config → Wi-Fi → WiFi IRAM speed optimization

2.2.8 Wi-Fi PHY Layer Packet Reception An optional feature of ESP32-C6, during light sleep, the PHY layer automatically receives packets, Wi-Fi no longer needs CPU participation to receive beacons, only Wi-Fi MAC + Baseband + RF are working, to save power consumption.

Note: Supplementary content: T/RX waveform diagram under different modes

(То	p) → (Compone	ent confi	ig → Wi	i-Fi				
	11111	11111111	<u>^</u> ^^		Espressif	IoT	Development	Framework	Configuration
	F	FTM Res	sponder s	support	t (NEW)				
*	Power	r Manag	gement fo	or stat	tion at dis	scon	nected		
[]	WiFi	GCMP S	Support((GCMP128	3 and GCMP2	256)			
[]] WiFi GMAC Support(GMAC128 and GMAC256)								
[*]] WiFi SoftAP Support								
[*]	WiFi	modem	automati	ically	receives t	the l	beacon		
[*]	Wifi	sleep	optimize	e when	beacon los	st			
	0 0		T.	NT					

• Configuration Item Name: - ESP_WIFI_ENHANCED_LIGHT_SLEEP - ESP_WIFI_EXTRA_IRAM_OPT - ESP_WIFI_RX_IRAM_OPT

• Configuration Path: (Top) \rightarrow Component config \rightarrow Wi-Fi \rightarrow WiFi modem automatically receives the beacon

3. Supplementary Content

3.1. The Difference Between DTIM and Listen Interval When the chip establishes a connection as a STA with the router, it will be informed of the router's DTIM, that is, how often the router will send a beacon. The STA needs to wake up at this time to receive the beacon and check if there is information that needs to be processed. The listen interval, on the other hand, is the STA telling the router how often it needs to wake up to receive the beacon, that is, the former is determined by the router, and the latter can be configured into the chip. The time interval is the value of DTIM or listen interval * 100 MS. If DTIM=10, the wake-up time interval is 10 * 100 ms = 1 s.

3.2. External 32KHz Crystal Using an external 32KHz crystal can achieve lower power consumption. The main reasons are as follows:

- The internal crystal is easily interfered with, while the external crystal has higher precision and can be used in various sleep situations.
- For applications that require high time accuracy, such as Bluetooth and Wi-Fi keep-alive, it is necessary to wake up regularly to receive beacons. Once the clock drifts too much and misses the reception point, it will cause the RF waiting window to lengthen, thereby greatly increasing power consumption.

Enabling the external clock source requires the following configuration:



- Configuration Item Name: ESP32C3_RTC_CLK_SRC_EXT_CRYS ESP_WIFI_EXTRA_IRAM_OPT ESP_WIFI_RX_IRAM_OPT
- Configuration Path: (Top) → Component config → ESP32XX-Specific → RTC clock source

There are two types of external crystals:

- External 32kHz crystal: This is an external passive crystal, which is the crystal recommended most of the time
- External 32kHz oscillator at 32K_XP pin: This is an external active crystal, which is more expensive and will cause the base current to rise by $50 \sim 100 \,\mu\text{A}$

The ESP32-C2 only supports an external active crystal oscillator as it does not have an internal passive crystal oscillator circuit. The layout of the crystal oscillator can be referred to the hardware design guide corresponding to each chip.

3.3. T/RX waveform under different modes Modem Mode (Physical layer receives beacon): The time cost of receiving a packet once is about 2.1 ms, and the current of receiving a packet is about 64 mA ~ 70 mA.



Active Mode RX: The duration of receiving a packet once is about 5.3 ms, and the current of receiving a packet is about 80 mA.



RX without receiving a packet: Compared with the situation where no packet is received, there will be no current waveform lasting for a period of time.



Active Mode TX: The duration is uncertain, and the TX current is related to the Wi-Fi channel.



ESP Wireless Transmission Power Configuration

This guide introduces how to set up the transmission power (TX Power) for Wi-Fi, BLE, Classic Bluetooth, Thread, and Zigbee in ESP-IDF.

Wi-Fi TX Power In ESP-IDF, the Wi-Fi transmission power can be set using the *esp_wifi_set_max_tx_power* function. The unit of transmission power is 0.25 dBm, so the value passed is four times the actual power. For example, when setting to 20 dBm, the value should be 80.

```
#include "esp_wifi.h"
void set_wifi_tx_power() {
    int8_t max_tx_power = 80; // 20 dBm
    esp_wifi_set_max_tx_power(max_tx_power);
}
```

If you want to use the default maximum power, you can also configure the maximum Wi-Fi transmit power in *menuconfig*.

menuconfig path: *Component config -> PHY -> Max WiFi TX power (dBm)*

BLE TX Power The BLE transmission power in ESP-IDF can be manually adjusted through the *esp_ble_tx_power_set* function. This function allows for setting different transmission powers for different types of transmission, such as broadcasting, scanning, and connection modes. The BLE power levels of different ESP chips may not necessarily be the same. For instance, in the case of ESP32, the power level range of BLE is from *ESP_PWR_LVL_N12* to *ESP_PWR_LVL_P9*.

The following example shows how to set the default transmit power of BLE to the maximum 9 dBm using ESP32:

```
#include "esp_gap_ble_api.h"
void set_ble_tx_power() {
    esp_ble_tx_power_set(ESP_BLE_PWR_TYPE_DEFAULT, ESP_PWR_LVL_P9); // Set maximum_
    $
    g dBm power
}
```

Example of ESP32 BLE Transmission Power Levels:

- ESP_PWR_LVL_N12: -12 dBm
- *ESP_PWR_LVL_N9*: -9 dBm
- *ESP_PWR_LVL_N6*: -6 dBm
- ESP_PWR_LVL_N3: -3 dBm
- ESP_PWR_LVL_P0: 0 dBm
- ESP_PWR_LVL_P3: 3 dBm
- *ESP_PWR_LVL_P6*: 6 dBm
- ESP_PWR_LVL_P9: 9 dBm (maximum power)

Examples of Some BLE Transmit Power Types:

- *ESP_BLE_PWR_TYPE_CONN_HDL0*: Transmit power of the first connection
- ESP_BLE_PWR_TYPE_CONN_HDL1: Transmit power of the second connection
- *ESP_BLE_PWR_TYPE_ADV*: Transmit power of broadcasting
- *ESP_BLE_PWR_TYPE_SCAN*: Transmit power of scanning
- ESP_BLE_PWR_TYPE_DEFAULT: Default transmit power

Some ESP chips can also configure the maximum BLE transmit power in *menuconfig*.

menuconfig path: Component config \rightarrow Bluetooth \rightarrow Controller Options \rightarrow BLE default Tx power level

In addition, for the new generation of ESP chips that support BLE (such as ESP32-C2, ESP32-C5, ESP32-C6, ESP32-H2), it is recommended to use the *esp_ble_tx_power_set_enhanced* API to adjust the BLE transmit power.

Classic Bluetooth TX Power The transmission power of Classic Bluetooth can be set through the *esp_bredr_tx_power_set* API, sharing the same setting method with BLE. In Classic Bluetooth, the transmission power is mainly used under the *ESP_BT_PWR_TYPE_DEFAULT* mode.

Here is an example:

```
#include "esp_bt.h"
void set_classic_bt_tx_power() {
    esp_bredr_tx_power_set(ESP_PWR_LVL_N0, ESP_PWR_LVL_P3);
}
```

Thread TX Power

- If developing based on Thread, it is recommended to adjust the transmit power of Thread through the openthread API *otPlatRadioSetTransmitPower*.
- If developing directly based on the 802.15.4 MAC driver, it is recommended to adjust the transmission power of Thread through *esp_ieee802154_set_txpower*.

Zigbee TX Power

• If developing based on Zigbee, it is recommended to adjust the transmission power of Zigbee through *esp_zb_set_tx_power*.

Notes

- Wi-Fi and BLE transmission power: When adjusting the transmission power of Wi-Fi and BLE using *esp_wifi_set_max_tx_power* and *esp_ble_tx_power_set*, it may be necessary to balance according to application requirements, as they share some hardware resources.
- Limitations of Menuconfig settings: *ESP_PHY_MAX_WIFI_TX_POWER* can only control the maximum transmission power of Wi-Fi, and has no direct impact on the transmission power of BLE and Classic Bluetooth.

ESP Memory Usage Optimization

Introduction Due to the hardware resource constraints of embedded chips, running complex application code may lead to insufficient memory. Most of the default configuration items in ESP-IDF prioritize performance, but in actual products, a good balance between performance and memory usage can be achieved through appropriate memory configuration.

Warning: Some memory optimization methods listed in this document may reduce system performance and stability. Sufficient performance and stability testing should be conducted after optimization to ensure the application meets its requirements.

Obtaining Current Free Memory Before optimizing memory usage, it is necessary to know the current memory consumption. For static memory usage, you can use the *idf.py size* and *idf.py size-components* commands to gather statistics memory usage.

For runtime memory consumption, *xPortGetFreeHeapSize()* and *xPortGetMinimumEverFreeHeapSize()* functions can be used to obtain the current free memory and the minimum heap memory remaining since system startup, respectively.

For more granular memory allocation, you can enable the *CONFIG_FREERTOS_USE_TRACE_FACILITY* and *CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS* options and then use the *vTaskList()* function to get the high watermark for each task stack.

Available Optimization Options

Note: The following optimization items are referenced from the ESP-IDF Programming Guide: Memory Optimization, which introduces the general memory optimization items for ESP.

1. Wi-Fi Memory Optimization

Wi-Fi performance is affected by many parameters, with trade-offs between them. Proper configuration can not only improve performance but also increase available memory and system stability. Configuration options vary significantly depending on the chip, so for Wi-Fi memory optimization and throughput, please refer to the relevant chip documentation under How to Improve Wi-Fi Performance.

2. LwIP Memory Optimization

Since RAM is allocated dynamically from the heap, most of lwIP's RAM usage is also dynamically allocated. Therefore, changing lwIP settings to reduce RAM usage might not affect idle RAM usage but can reduce peak RAM consumption. For LwIP memory optimization, see LwIP Minimum RAM Usage.

3. Mbed TLS Memory Optimization

Mbed TLS is a C library used to implement cryptographic primitives, X.509 certificate handling, and SSL/TLS and DTLS protocols. When using the Mbed library for encryption and decryption, memory usage can be reduced by adjusting its configuration. For Mbed memory optimization, see Mbed Minimum RAM Usage.

4. BLE Memory Optimization

Compared to Bluedroid, NimBLE uses less memory and a smaller firmware size. Additionally, you can disable certain features and reduce buffer sizes to achieve optimal memory usage. For scenarios where BLE is only used for provisioning, try the following configurations:

Reduce maximum connections and related parameters:

- CONFIG_BT_NIMBLE_MAX_CONNECTIONS=1
- CONFIG_BT_NIMBLE_MAX_BONDS=2
- CONFIG_BT_NIMBLE_MAX_CCCDS=2
- CONFIG_BT_NIMBLE_WHITELIST_SIZE=2

If there are no high-speed requirements, reduce buffer sizes and counts:

- CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT=12
- CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE=100
- CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT=4
- CONFIG_BT_NIMBLE_TRANSPORT_ACL_FROM_LL_COUNT=12
- CONFIG_BT_NIMBLE_TRANSPORT_EVT_COUNT=5
- CONFIG_BT_NIMBLE_TRANSPORT_EVT_DISCARD_COUNT=3

Disable unnecessary features:

- CONFIG_BT_NIMBLE_ROLE_CENTRAL=n
- CONFIG_BT_NIMBLE_ROLE_OBSERVER=n
- CONFIG_BT_NIMBLE_SECURITY_ENABLE=n

5. Other Configuration Optimizations

For performance and power-saving purposes, some functions are placed in IRAM by default. If needed, you can reconfigure these functions to be placed back into Flash:

- CONFIG_SPI_SLAVE_ISR_IN_IRAM=n
- CONFIG ESP WIFI IRAM OPT=n
- CONFIG_ESP_WIFI_RX_IRAM_OPT=n
- CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH=y
- CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH=y
- CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH=y

Reduce the length of some system queues:

• CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE=16

Enable nano format options:

• *NEWLIB_NANO_FORMAT=y*

Disable unnecessary features:

- CONFIG_WS_TRANSPORT=n
- 6. Reduce Task Stack Sizes

In the default ESP-IDF configuration, Task Stack sizes include a large margin to accommodate different chips and allow users to add application code. Reducing Task Stack sizes can free up memory for dynamic allocation.

Common Task Stack configuration items:

- CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE
- CONFIG_BT_LE_CONTROLLER_TASK_STACK_SIZE
- CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE
- CONFIG_ESP_TIMER_TASK_STACK_SIZE
- CONFIG_FREERTOS_IDLE_TASK_STACKSIZE
- CONFIG_LWIP_TCPIP_TASK_STACK_SIZE

Optimization Example and Result Statistics Testing the bleprph_wifi_coex example demonstrates the effect of memory optimization by comparing memory usage before and after Wi-Fi and BLE connections.

1. Hardware and Software Environment

- Hardware: ESP32-C2
- Software: ESP-IDF 11eaf41b37267ad7709c0899c284e3683d2f0b5e (tag: v5.2)
- Example: examples/bluetooth/nimble/bleprph_wifi_coex

2. Configuration Changes

- CONFIG_PARTITION_TABLE_SINGLE_APP_LARGE=y
- CONFIG_EXAMPLE_ESP_PING_COUNT=5
- CONFIG_BT_ENABLED=y
- *CONFIG_BT_NIMBLE_ENABLED=y*
- CONFIG_BT_NIMBLE_MAX_BONDS=2
- CONFIG_BT_NIMBLE_MAX_CCCDS=2
- CONFIG_BT_NIMBLE_HOST_TASK_STACK_SIZE=2048
- CONFIG_BT_NIMBLE_ROLE_CENTRAL=n
- CONFIG_BT_NIMBLE_ROLE_OBSERVER=n
- CONFIG_BT_NIMBLE_SECURITY_ENABLE=n
- CONFIG_BT_NIMBLE_MSYS_1_BLOCK_COUNT=12
- CONFIG_BT_NIMBLE_MSYS_1_BLOCK_SIZE=100
- CONFIG_BT_NIMBLE_MSYS_2_BLOCK_COUNT=4
- CONFIG_BT_NIMBLE_TRANSPORT_ACL_FROM_LL_COUNT=12
- CONFIG_BT_NIMBLE_TRANSPORT_EVT_COUNT=5
- CONFIG_BT_NIMBLE_TRANSPORT_EVT_DISCARD_COUNT=3
- CONFIG_BT_NIMBLE_50_FEATURE_SUPPORT=n
- CONFIG_BT_NIMBLE_WHITELIST_SIZE=2
- CONFIG_BT_LE_CONTROLLER_TASK_STACK_SIZE=2048
- CONFIG_ESP_COEX_SW_COEXIST_ENABLE=n
- CONFIG_ESP_EVENT_POST_FROM_ISR=n
- CONFIG_ESP_HTTP_CLIENT_ENABLE_HTTPS=n
- CONFIG_RINGBUF_PLACE_FUNCTIONS_INTO_FLASH=y
- CONFIG_RINGBUF_PLACE_ISR_FUNCTIONS_INTO_FLASH=y
- CONFIG_ESP_SYSTEM_EVENT_QUEUE_SIZE=16
- CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE=1536
- CONFIG_ESP_TIMER_TASK_STACK_SIZE=2048
- CONFIG_ESP_WIFI_STATIC_RX_BUFFER_NUM=3
- CONFIG_ESP_WIFI_DYNAMIC_RX_BUFFER_NUM=6
- CONFIG_ESP_WIFI_DYNAMIC_TX_BUFFER_NUM=6
- CONFIG_ESP_WIFI_AMPDU_TX_ENABLED=n
- CONFIG_ESP_WIFI_AMPDU_RX_ENABLED=n
- CONFIG_ESP_WIFI_IRAM_OPT=n
- CONFIG_ESP_WIFI_RX_IRAM_OPT=n
- CONFIG_ESP_WIFI_ENABLE_WPA3_SAE=n
- CONFIG ESP WIFI SOFTAP SUPPORT=n
- CONFIG_ESP_WIFI_SLP_BEACON_LOST_OPT=y
- CONFIG ESP WIFI ESPNOW MAX ENCRYPT NUM=0
- CONFIG ESP WIFI ENTERPRISE SUPPORT=n
- CONFIG FREERTOS IDLE TASK STACKSIZE=768
- CONFIG_FREERTOS_USE_TRACE_FACILITY=y
- CONFIG_FREERTOS_USE_STATS_FORMATTING_FUNCTIONS=y
- CONFIG_FREERTOS_PLACE_FUNCTIONS_INTO_FLASH=y
- CONFIG_LWIP_L2_TO_L3_COPY=y
- CONFIG_LWIP_MAX_SOCKETS=8
- CONFIG_LWIP_SO_LINGER=y
- CONFIG_LWIP_SO_RCVBUF=y
- CONFIG_LWIP_NETBUF_RECVINFO=y
- CONFIG_LWIP_GARP_TMR_INTERVAL=30
- CONFIG_LWIP_MLDV6_TMR_INTERVAL=30
- CONFIG_LWIP_TCPIP_RECVMBOX_SIZE=16
- CONFIG_LWIP_DHCP_RESTORE_LAST_IP=y

- CONFIG_LWIP_DHCPS=n
- CONFIG_LWIP_IPV6_NUM_ADDRESSES=6
- CONFIG_LWIP_IPV6_FORWARD=y
- CONFIG_LWIP_TCP_SND_BUF_DEFAULT=6144
- CONFIG_LWIP_TCP_SACK_OUT=y
- CONFIG_LWIP_MAX_UDP_PCBS=24
- CONFIG_LWIP_UDP_RECVMBOX_SIZE=12
- CONFIG_LWIP_TCPIP_TASK_STACK_SIZE=2048
- CONFIG_LWIP_IPV6_MEMP_NUM_ND6_QUEUE=6
- CONFIG_LWIP_IPV6_ND6_NUM_NEIGHBORS=8
- CONFIG_MBEDTLS_SSL_IN_CONTENT_LEN=6288
- CONFIG_MBEDTLS_SSL_OUT_CONTENT_LEN=2048
- CONFIG_MBEDTLS_DYNAMIC_BUFFER=y
- CONFIG_MBEDTLS_DYNAMIC_FREE_CONFIG_DATA=y
- CONFIG_MBEDTLS_SSL_KEEP_PEER_CERTIFICATE=n
- CONFIG_MBEDTLS_CERTIFICATE_BUNDLE=n
- CONFIG_NEWLIB_TIME_SYSCALL_USE_HRT=y

3. **Optimization Results**

The table below shows the system's available memory before and after optimization. The results indicate that approximately 60 KB of memory can be saved through configuration optimization.

Available Memory	Before Optimiza-	After Optimization	Difference
	tion		
Current Available (bytes)	33,232	95,040	61,808
Minimum Available (bytes)	25,804	91,252	65,448

2.2.4 Network Protocol Related

This section summarizes the usage experience related to network protocols.

Mbed TLS Common Error Troubleshooting

The following are some common error logs in ESP-IDF's Mbed TLS, their corresponding causes, and possible solutions.

Example: esp-tls: couldn' t get hostname for :drive.google.com: getaddrinfo() returns 202, addrinfo=0x0 The complete log for this issue is often:

```
E (83792) esp-tls: couldn't get hostname for :drive.google.com: getaddrinfo()_

→returns 202, addrinfo=0x0

E (83792) esp-tls: Failed to open new connection

E (83792) transport_base: Failed to open a new connection

E (83802) HTTP_CLIENT: Connection failed, sock < 0</pre>
```

Error analysis:

• *getaddrinfo()* failed to resolve, returning error code 202, indicating that *drive.google.com* could not be resolved, and *addrinfo=0x0* indicates that *getaddrinfo()* did not return a valid IP address.

Possible reasons:

• DNS resolution failed, possibly because the DNS server did not respond, or ESP did not receive this response.

Warning: If multiple Netif interfaces are used simultaneously, such as Wi-Fi and LTE, since lwIP cannot support each Netif having an independent DNS server domain, if the DNS server can only be used by a certain Netif interface, this error will also occur when the DNS is overwritten by other Netif interfaces.

Solution:

• It is best to enable the DNS debug log of the lwIP layer, or further confirm the reason through wireless packet capture.

Note:

- 1. The method to enable the debug log of the lwIP layer can refer to this ESP-FAQ.
- 2. The method of wireless packet capture can refer to Espressif Wireshark User Guide.

Example: esp-tls-mbedtls: mbedtls_ssl_handshake returned -0x7F00 mbedtls -0x7F00

The complete log for this issue is often:

```
E (3968) esp-tls-mbedtls: mbedtls_ssl_handshake returned -0x7F00
E (3968) esp-tls: Failed to open new connection
E (3968) TRANSPORT_BASE: Failed to open a new connection
E (3978) HTTP_CLIENT: Connection failed, sock < 0
E (3978) http_post: http post request failed: ESP_ERR_HTTP_CONNECT
I (3988) http_event_handler: HTTP_EVENT_DISCONNECTED
W (3998) http_event_handler: last esp error code: 0x801a, mbedtls failure: 0x2880</pre>
```

Error Analysis:

• *mbedtls_ssl_handshake* returns error code -0x7F00. By querying Mbed TLS Error Codes, it can be found that the cause is MBEDTLS_ERR_SSL_ALLOC_FAILED. Insufficient ESP available memory leads to Mbed TLS memory allocation failure.

Solution:

• Optimize ESP memory to ensure that the maximum free block of ESP is sufficient for allocation to Mbed TLS.

Such problems can often be referred to Mbed TLS Error Codes for some references. Common error codes are summarized as follows:

Error Name	Error Code	Cause of Error	Solution
MBEDTLS_ERR_SSL_ALLC	C <u>-(</u> FxATHOBD	Memory allocation failed	Check memory usage, ensure
			sufficient available memory
MBEDTLS_ERR_X509_FAT	AL <u>O</u> ÆRROR	Certificate parsing failed	Check the certificate file, en-
			sure the certificate format is
			correct
MBEDTLS_ERR_X509_CER	T_0%12R010Y_FA	ILED tificate verification failed	Ensure the core CA certificate
			is valid and check the certifi-
			cate chain
MBEDILS_ERR_X509_BAL	_ HN#280 0DATA	Invalid certificate input data	Ensure the provided certifi-
MDEDTLC EDD SSL CONN		Commonstian accession d EOE	Check the servestion status
MBEDILS_EKK_SSL_CONT	-1001280	Connection received EOF	Check the connection status,
			ensure the other end has not
MDEDTLS EDD SSL DECE		COLONIARICKET	Ensure the other and is config
MDEDILS_ERK_SSL_KECE	1 4 6 1 7 1 6 1 6 1 7 5 1	LSISTECEN <u>v</u> ential inervisession ticket	urad to support session tickets
MREDTIS ERR SSI INTE		Internal error	Ensure the Mbed TLS library
MDEDTES_EKK_SSE_INTE			is configured correctly and en-
			able debug to view detailed
			logs
MBEDTLS ERR SSL TIME	010166800	Operation timeout	Increase the timeout or check
			the network condition
MBEDTLS ERR SSL PEER	GOX78EONOTI	FY he other end notifies the con-	Close the connection and re-
		nection to close	establish the session
MBEDTLS_ERR_SSL_NO_C	L i@x7# 80ERTI	FICIAERE certificate required but	Ensure the client has provided
		not provided	the correct certificate and con-
			figured two-way authentica-
			tion
MBEDTLS_ERR_SSL_INVA	LHO <u>x</u> RECOORD	Invalid TLS record	Ensure the communi-
			cation data is complete
			and reliable, check if
			the problem is caused by
			IN_CONTENT_LENGTH
			being too small (for example,
			is it 16 K). Also, confirm that
			the TLS protocol parameters
			configured by the other end
MDEDTIC EDD X500 FEA			are consistent with this end.
MBEDILS_ERK_X509_FEA	I GUKKE <u>U</u> KUNAVA		Ensure that the relevant fea-
			fouring the corresponding on
			tions if support for a spe
			cific encryption algorithm is
			required
MBEDTLS FRR X509 INV		Invalid certificate format	Ensure that the certificate file
		invalid contineate format	is in the correct format and
			try to use the PEM format as
			much as possible
1	1	1	

Table 4: Common Mbed TLS Error Codes

In addition, ESP-IDF also encapsulates some Mbed TLS APIs and summarizes some ESP TLS return values, which can be referred to ESP TLS return values.

Common Socket Error Troubleshooting

The following summarizes common error logs in ESP-IDF Socket, their corresponding causes, and possible solutions.

Example: transport_base: poll_read select error 113, errno = Software caused connection abort, fd = 56 Error Message Analysis:

- poll_read select error 113
- *errno* = 113 (Software caused connection abort)
- fd = 56

Possible Causes:

• 113 is usually due to TCP keepalive heartbeat timeout, retransmission timeout causing abnormal link disconnection.

Solutions:

- Check for abnormal logs on the server side.
- Ensure stable network connection.
- Implement heartbeat mechanism at the application layer to avoid disconnection due to long periods of no data communication.

Note: When analyzing TCP flow abnormal issues, you can use the debug patch script to print TCP sequence number and other information to help locate the problem.

Usage: Enter the ESP-IDF main directory in the terminal and execute the command python path to
at_net_debug.py> to apply this patch. After applying the patch, subsequent ESP logs will include TCP sequence number (seq) and acknowledgment number (ack) and other debugging information.

Common errno and their analysis

Errno Defini- tion	Cor- re- spond- ing	Corre- sponding ERR	Corresponding Explanation	Cause	Localization Mea- sures
ENOMEM	Value 12	ERR_MEM	Out of memory er- ror	Failed to allocate mem- ory or mailbox is full, msg sending failed	Print remaining memory and min- imum memory, check if there is insufficient memory
ENOBUFS	105	ERR_BUF	Buffer error	1. Insufficient space, the header length has exceeded the allocated length 2. socket failed to allocate netconn 3. option len too long	Add log for posi- tioning
EWOULD- BLOCK	11	ERR_TIME / ERR_WOU	OTilifieout / Opera- tion would block LDBLOCK	1. Timeout –Receive Timeout (SO_RCVTIMEO set) 2. Operation would block – too fast send/receive calls under non-blocking 3. Operation would block – Send timeout (SO_SNDTIMEO set)	Add print at ERR_TIMEOUT; after nonblock- ing, connec- tion/sending/receiving are immediately returned, the appli- cation layer should handle it well
EHOSTUN- REACH	118	ERR_RTE	Routing problem	Can' t find the outgoing route (netif)	Add log for posi- tioning, check if there is a satisfying routing interface for the destination IP
EIN- PROGRESS	119	ERR_INPR	O Opdfass on in progress	The operation is in progress, non-blocking mode connect will en- counter, DNS query will also encounter	Normal errno
EINVAL	22	ERR_VAL	Illegal value	Parameter error (param- eters set by option, fd passed in by select/poll)	Check if there is a problem with the parameters of the function call
EADDRI- NUSE	112	ERR_USE	Address in use	The address or port is al- ready bound when bind- ing	Check if the bound address or port is already bound, can be solved with SO_REUSEADDR
EALREADY	120	ERR_ALRE	ADA ady connecting	The socket is already con- necting or is in the listen state, and the application layer calls connect or lis- ten again	Check the code logic
EISCONN	127	ERR_ISCO	NNOnn already estab- lished	The socket is already in Connected	Check the code logic
ENOTCONN	128	ERR_CON	Not connected / Connection closed	The socket is in an uncon- nected state or the other	Generally, the other end has
Espressif Systems	\$	ERR_CLSD	125 Submit Document	end has disconnected FIN Feedback	discon Release master we are still using this socket to send data, confirm by

Summary

- Through the *errno* code, the root cause of the socket problem can be quickly located.
- Combined with log analysis, error handling can be optimized to improve system stability.
- Use *getsockopt()* to read *SO_ERROR* to get more error information.

Hope this information can help troubleshoot ESP-IDF Socket related issues.

2.2.5 System Related

This section summarizes the usage experience related to the ESP system.

Memory Usage Comparison

This document is used to summarize the memory consumption when running common examples on different chips.

Comparison of Remaining Memory Size for Common Examples Please note that the following test results have not undergone any memory optimization, they are simply the results obtained after directly running the basic examples in ESP-IDF. If you want to see the results after memory optimization, you can refer to next part, **Example: Comparison of remaining memory size before and after memory optimization for ESP32-C3**.

			-				
ex-	ESP32	ESP32-	ESP32-	ESP32-	ESP32-	ESP32-	ESP32-
am-		C3	C3(ECO7)	C2	S3	C6	H2
ple							
Empty	305 K	330 K	330 K	196 K	389 K	465 K	264 K
project							
Sta-	229 K	222 K	230 K	104 K	282 K	327 K	not support
tion							
Gatt_set	rv205 K	218 K	230 K	81 K	280 K	365 K	152 K
bleprph	209 K	219 K	233 K	100 K	281 K	379 K	170 K
provi-	164 K	134 K	152 K	22 K	193 K	262 K	not support
sion(ble							
&							
Wi-Fi							
co-							
exis)							

 Table 5: Comparison of Application Firmware Size

Note: This data was tested in IDF v5.3.1, using the default sdkconfig for each example.

Note: In ESP32-C3 (ECO7), there is more available RAM for wireless applications: 10 KB for Wi-Fi related, 15 KB for BLE, and 20 KB for Wi-Fi & BLE coexistence. If you want to use ESP32-C3 (ECO7), please enable the following settings in menuconfig:

- Component config > Hardware Settings > Chip revision > Minimum Supported ESP32-C3 Revision > Rev 1.1
- Component config > SPI Flash driver > Use esp_flash implementation in ROM

Example:	Comparison of	remaining memory	size before and	l after memory	optimization	for ESP32-C3
----------	----------------------	------------------	-----------------	----------------	---------------------	--------------

example	Before memory optimization	After memory optimization
Physical memory	400 K	400 K
Station	230 K	304 K
Station + 1TLS(MQTTS)	210 K	300 K
bleprph_wifi_coex	178 K	248 K

Table 6: Available free heap after optimization for ESP32-C3/ESP32-C2

Note:

- ESP32-C3 data tested in ESP-IDF v5.2.2
- Print the station' s free heap memory after the *got ip* event
- Station + 1TLS(MQTTS) data is based on the ESP-IDF mqtt/ssl example, print the free heap memory after the *MQTT_EVENT_DATA* event
- In the ESP-IDF bleprph_wifi_coex example test, nimble BLE broadcast starts after the *got ip* event, then print the free heap memory

System Clock

Clock Overview The general functions of a clock are to synchronize clock signals and timing. It can be simply divided into:

- High-performance clock
- Low-power clock

High-performance Clock The high-performance clock is used to provide the working clock for the CPU and digital peripherals. It is currently divided into the following two types:

- PLL_CLK: Provides a high-frequency internal clock of 320 MHz or 480 MHz
- XTAL_CLK: Provides a stable 40 MHz external crystal clock

XTAL_CLK can provide a reference clock for PLL_CLK, and PLL_CLK (Phase-Locked Loop) can lock the phase of the input signal and generate an output signal that is an integer multiple of the input frequency, to achieve the clock signal output of the frequency-doubled signal.

Low-power Clock The low-power clock is the clock source for the RTC module and low-power peripherals. It is currently divided into the following three types:

- XTAL32K_CLK: Provides a stable 32 KHz external crystal clock, suitable for applications that require precise timing
- FOSC_CLK: Provides a fast adjustable 17.5 MHz internal RC oscillator, suitable for high-speed operations
- RTC_CLK: Provides a slow adjustable 136 KHz RC oscillator, very suitable for low-power timing functions

Comparison of Different Clocks

Clock	Advantages	Disadvantages
Source		
PLL	Very precise clock frequency. Can provide a very	High power consumption
	high clock frequency	
XTAL	Very precise clock frequency, lower power con-	Fixed frequency, long startup time (100 us), usu-
	sumption	ally an external crystal
RC Os-	Very low power consumption, short startup time	Lower accuracy, and easily affected by the envi-
cillator		ronment

For clock-related details of each chip, please refer to the corresponding technical reference manual, such as the clock section in the ESP32-C3 Technical Reference Manual.

Time Acquisition and Calibration

Introduction This section has been systematically narrated in the System Time of the ESP-IDF Programming Guide, and this document is only a supplementary summary.

Common Time Acquisition Interfaces

- esp_timer_get_time
 - Get the time since esp_timer initialization, in µs
 - Use RTC for timing when the ESP chip is in sleep mode, and compensate for time through RTC after exiting sleep mode
- esp_rtc_get_time_us
 - Get the time of the RTC counter (RTC_SLOW_CLK), in µs
- esp_cpu_get_cycle_count
 - First get the CPU cycle count, then divide by the result corresponding to *esp_rom_get_cpu_ticks_per_us()* to get the corresponding time, in µs
- gettimeofday
 - Use esp_timer as the clock source, in µs
 - Use RTC for timing when the ESP chip is in sleep mode, and compensate for time through RTC after exiting sleep mode
- xTaskGetTickCount
 - Get the FreeRTOS Tick count, in Ticks

Here are some experience summaries:

- To get the current time, it is recommended to use the POSIX function *gettimeofday*, but this function has a slightly higher overhead
- Using *esp_timer_get_time* can generate timestamps with microsecond precision, but each call to the timing function will generate a certain amount of overhead
- When measuring small code segments that run for less than 1-2 ms, there may be large differences in timing measurements due to the function being in flash. This problem can be solved by moving the code to IRAM
- By using GPIO to quickly flip, very small code execution times can be observed through a logic analyzer

RC Clock Calibration There are many sources of RTC clock, but only the external 32K clock can provide a higher precision clock, while other RC clocks have lower precision and will accumulate more errors after running for a long time. There is no significant difference in the precision of different clock sources inside the RTC.

Watchdog

ESP Watchdog Timer Classification Taking ESP32-C3 as an example, as follows:

• ESP32-C3

– Digital Watchdog Timer

- * Main System Watchdog Timer (MWDT0 & MWDT1)
- * RTC Watchdog Timer (RWDT)
- Analog Watchdog Timer
 - * Super Watchdog (SWD)

Watchdog Trigger Principle Taking ESP32-C3 as an example, it has two main system watchdog timers, namely MWDT0 and MWDT1. The digital watchdog will go through multiple stages during operation, and each stage can configure separate timeout time and timeout action. The existing logic is as follows:

The task watchdog uses MWDT0, the interrupt watchdog uses MWDT1, if ESP does not feed the dog in time, causing the watchdog timeout will trigger the watchdog interrupt.

Note: ESP32-C2 only has one timer group, so there is only one main system watchdog MWDT0, which is bound to the interrupt watchdog. At this time, the task watchdog is implemented using esp_timer.

Interrupt Watchdog The interrupt watchdog is mainly used to detect scenarios where FreeRTOS cannot schedule tasks for a long time. In a real-time operating system, the task scheduling mechanism ensures that high-priority tasks that need real-time processing can quickly get the opportunity to execute, thereby avoiding long delays. If the system cannot normally schedule tasks, some critical tasks (such as Wi-Fi packet transmission and reception processing) may experience anomalies.

Working Principle The interrupt watchdog uses the hardware watchdog timer of Timer Group 1, monitoring the system task scheduling status by adding a feed-dog operation in the SysTick interrupt. The SysTick interrupt is by default the lowest interrupt priority 1 in ESP-IDF, while the interrupt priority of the interrupt watchdog timer is 4, higher than the interrupt levels of most peripherals, and cannot be easily shielded by the system.

When the watchdog is not fed in the SysTick interrupt for a long time (default 300 ms), the interrupt watchdog interrupt will be triggered, the system will enter the interrupt watchdog exception handler, print register information, and crash.

Note: The interrupt disable operation will only turn off interrupts of priority 3 and below. In the Xtensa instruction set, interrupts of priority 4 and above can only be executed in assembly functions, therefore the interrupt watchdog cannot be masked by regular interrupt disable operations.

SysTick Timer (SysTick) The SysTick timer is a system tick timer provided by the chip, featuring automatic reload and overflow interrupt functions. In RTOS, SysTick provides the necessary clock ticks for task scheduling, acting as the system's "heartbeat". Whenever a SysTick interrupt is triggered, the system performs a task switch.

The overflow interrupt interval can be adjusted by changing the tick value through the configuration item Component config -> FreeRTOS -> Tick rate. The default value is 100, which means a SysTick interrupt is triggered every 10 ms.

Warning:

- Too frequent SysTick interrupts can lead to frequent task scheduling, reducing system efficiency.
- A too slow SysTick interrupt may cause task response delay
- All time-related behaviors of the operating system are based on Tick, such as vTaskDelay(100) delays 1 s when the Tick rate is 100, and only delays 100 ms when the Tick rate is 1000.



Fig. 1: Interrupt Watchdog Working Principle Diagram

Trigger Reason The fundamental reason for the triggering of the interrupt watchdog is that the SysTick interrupt cannot be executed normally, which mainly includes the following three situations:

1. Long-term Interruption Shutdown

The operation of disabling interrupts is used to protect program segments that cannot be interrupted midway, preventing the system from responding to interrupt requests. Long-term disabling of interrupts is the main cause of triggering the interrupt watchdog.

Common scenarios include:

- Enter critical section (portENTER_CRITICAL)
- Use spinlock
- Other code segments that require interrupt protection

2. Blockage Exists in the Interrupt Service Routine

The Interrupt Service Routine (ISR) should adhere to the principle of rapid processing, transferring timeconsuming operations to a non-interrupt environment for handling. If there is complex logic or blocking operations within the ISR, and it fails to exit before the interrupt watchdog times out, an exception will be triggered.

Frequently Asked Questions:

- The ISR contains an infinite loop
- Perform time-consuming operations in ISR
- Call blocking function in ISR

3. Interrupt Flag Not Cleared

After the ISR (Interrupt Service Routine) is executed, the corresponding interrupt flag needs to be cleared. If it is not cleared correctly, the interrupt will continue to trigger, preventing the normal execution of the SysTick interrupt.

Note: The system often automatically clears the corresponding interrupt flag bit, so manual operation by the user is not required.

Configuration Options The configuration related to the interrupt watchdog is located at Component config → ESP System Settings:

- Interrupt watchdog: Enable or disable the interrupt watchdog function.
- Interrupt watchdog timeout (ms): Interrupt watchdog timeout duration, default is 300 ms.

Problem Analysis Method When a watchdog interrupt exception is triggered, the system will stay at the location where the problem occurred. Analysis can be carried out in the following ways:

- 1. View Program Counter (PC) and Call Stack (Backtrace): Locate the problem code position
- 2. Check the context identifier in the crash information:
 - Includes Core X was running in ISR context indicates the problem occurred in the interrupt function
 - The absence of this information usually indicates an issue caused by an interrupt, and the specific reason needs to be analyzed based on the location of the problem.

Practical Case Study Analysis Issues with interrupt watchdog often combine with the FreeRTOS system in practical applications, manifesting in various forms. For a detailed analysis of actual cases and solutions, please refer to the *Interrupt wdt timeout on CPU0/CPU1* section, which includes a complete problem analysis process and several typical examples.

Task Watchdog The Task Watchdog is used to detect situations where a specific task occupies CPU resources for a long time. In the FreeRTOS preemptive kernel, high-priority tasks always get CPU resources first, which may cause low-priority tasks to be unable to execute for a long time. The Task Watchdog monitors the execution status of key tasks to ensure the correct manifestation of the system' s multitasking features.

Working Principle The task watchdog uses the hardware watchdog timer of Timer Group 0. By default, the system determines whether a watchdog event has occurred by monitoring the IDLE task. The IDLE task is the lowest priority system task in FreeRTOS. If this task can run normally, it indicates that other tasks have not occupied CPU resources for a long time.

When the system enters the IDLE task, it resets the watchdog timer through the hook function of the IDLE task. If it fails to enter the IDLE task for a long time, the watchdog will trigger a timeout.

Monitoring Task Management In addition to the default IDLE task monitoring, other tasks can also be added as monitoring targets:

- Use <code>esp_task_wdt_add(NULL)</code> to add the current task to the monitoring list.
- The monitored task must periodically call <code>esp_task_wdt_reset()</code> for watchdog feeding operations.
- The system requires all monitored tasks to complete the watchdog feeding before it will reset the watchdog timer.

Configuration Options The configuration related to the task watchdog is located at Component config → ESP System Settings:

- Initialize Task Watchdog Timer on startup: Enable or disable the task watchdog.
- Invoke panic handler on Task Watchdog timeout: Set whether to restart the system when the watchdog is triggered
- Task Watchdog timeout period (seconds): The timeout period for the task watchdog, default is 5 seconds.
- Watch CPU0 Idle Task: Monitor the IDLE task of CPU0 (In a dual-core system, you can choose whether to monitor CPU1)



Fig. 2: Task Watchdog Working Principle Diagram

Problem Diagnosis Basic Diagnosis

When the task watchdog is triggered, the system will print relevant information, including:

- The task currently running
- Relevant call stack information

Advanced Diagnostics

For complex applications, you can use the vTaskGetRunTimeStats() interface to analyze CPU usage:

- 1. Enable relevant configurations:
 - Component config -> FreeRTOS -> Enable FreeRTOS trace facility
 - Component config -> FreeRTOS -> Enable FreeRTOS stats formatting functions
 - Component config -> FreeRTOS -> Enable FreeRTOS to collect run time stats
- 2. Loop call to the statistical interface:

```
void run_time_monitor(void)
{
    char *pbuffer = (char *)calloc(1, 2048);
    printf("------\r\n");
    vTaskGetRunTimeStats(pbuffer);
    printf("%s", pbuffer);
    printf("------\r\n");
    free(pbuffer);
}
```

Note: When using the vTaskGetRunTimeStats interface, you need to disable the Invoke panic handler on Task Watchdog timeout configuration, otherwise the system will restart due to watchdog triggering, and it will not be able to print statistical information.

Solution Code Optimization Strategy

1. For tasks that indeed have issues:

- Use blocking interfaces with timeouts in the loop, such as xSemaphoreTake() (the timeout cannot be 0)
- Use vTaskDelay() to actively yield CPU resources
- Optimize algorithm, reduce single execution time
- 2. For normal high-load tasks (without wishing to reduce operational efficiency):
 - Manually add a watchdog operation in high-load tasks

Manual Watchdog Operation

For tasks that need to run for a long time, you can use the manual watchdog feeding method:

```
static void long_running_task(void* arg)
{
    // Remove watchdog monitoring of IDLE task
    esp_task_wdt_delete(xTaskGetIdleTaskHandleForCPU(0));
    // Add the current task to the monitoring list
    esp_task_wdt_add(NULL);
    while (1) {
        // Execute time-consuming operations
        // Feed the dog regularly
        esp_task_wdt_reset();
    }
    // Restore IDLE task monitoring when the task ends
    esp_task_wdt_delete(NULL);
    esp_task_wdt_add(xTaskGetIdleTaskHandleForCPU(0));
}
```

Warning: The manual watchdog feeding operation is relatively complex, as ESP-IDF requires all monitored tasks to complete the watchdog feeding before the watchdog timer is reset. It is not recommended to arbitrarily modify the internal implementation of ESP-IDF.

TimerGroup Watchdog Common LOG:

```
1. rst:0x8 (TG1WDT_SYS_RST),boot:0xc (SPI_FAST_FLASH_BOOT)
2. rst:0x7 (TG0WDT_SYS_RST),boot:0xc (SPI_FAST_FLASH_BOOT)
```

Trigger conditions:

• Task interrupts and task scheduling cannot be triggered normally, it can be considered that there is an exception in CPU operation

Possible reasons:

- CPU instruction fetching is abnormal, at this time suspect whether there is hardware interference in Flash or PSRAM communication, such as abnormal occupation of Flash or PSRAM pins, high frequency interference, etc.
- Unstable power supply
- Possible wild pointer problem

RTC Watchdog Common LOG:

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

Features:

- In addition to resetting digital peripherals, it also resets the RTC module
- Can be called in the program

Stages that can trigger the RTC watchdog:

- Boot stage: This watchdog needs to be enabled at this stage through the CON-FIG_BOOTLOADER_WDT_ENABLE option
- · Restart stage
- Panic Stage
- Enter the light sleep stage, after entering light sleep, the hardware will automatically turn off the RTC watchdog

Other watchdogs

External hardware watchdog The user may encounter the following abnormal scenarios:

- The hardware has unstable power supply
- Occasionally, the chip hangs without triggering any reset

In this case, it is recommended to connect an external hardware watchdog, which can be fed by periodically flipping the GPIO through the software program, to avoid the situation where the ESP chip is always hanging.

Timer

ESP Timer Classification

- Timer
- System Timer
 - * esp_timer
 - FreeRTOS Tick
- General Timer
 - * Independent Hardware Timer

The following will introduce each type of ESP timer one by one.

esp_timer Since there already exists an esp_timer document in the ESP-IDF programming guide, only some additional supplements will be made here.

The features of esp_timer are as follows:

- The clock source is XTAL_CLK
- Supports time compensation by loading the sleep time recorded by the RTC timer through the system timer after waking up from light-sleep
- Taking ESP32 as an example, ESP32 often runs at a CPU frequency of 240 MHz and uses task distribution to determine the minimum period of esp_timer. At this time, the timeout value configured by the one-time timer needs to be greater than 20 µs, and the timeout value of the periodic timer needs to be greater than 50 µs

There are two types of esp_timer callbacks:

• Task distribution method (default): Distribute timer callback functions from a single high-priority esp_timer task. This method is suitable for timer callback functions that do not have high timing requirements

• Interrupt distribution method: Directly distribute timer callback functions from the interrupt service routine. This method is suitable for simple, low-latency timer callback functions that only run for a few μ s. The advantage is that it can ensure that the delay between the event and the execution of the callback is short and is not affected by other tasks.

The way to configure as an interrupt distribution method is: Enable CON-FIG_ESP_TIMER_SUPPORTS_ISR_DISPATCH_METHOD in menuconfig.

GPTimer Since there already exists a GPTimer document in the ESP-IDF programming guide, only some additional supplements will be made here.

The features of GPTimer are as follows:

- The clock source is XTAL_CLK && APB_CLK
- Implemented based on timer group peripherals
- The application exclusively uses hardware resources and is not easily interfered with by other modules
- More functions, can be linked with other peripherals through ETM (only new chips such as ESP32-C6 support)

FreeRTOS Timer Since there already exists a FreeRTOS Timer API description in the ESP-IDF programming guide, only some additional supplements will be made here.

The features of the FreeRTOS Timer are as follows:

- The clock source is FreeRTOS Tick
- Pure software timer
- Uses a low-priority timer task (the priority can be configured through the CON-FIG_FREERTOS_TIMER_TASK_PRIORITY option), the maximum resolution is the FreeRTOS Tick cycle

Clock	Advantages	Disadvantages
esp_timer	 High precision and accuracy Easy to use 	The system also uses it, and when there is a blockage in the application layer, it will affect the system
GPTimer	 Highest precision and accuracy, can be used for waveform generation Provides additional features such as event capture 	Not convenient to use
FreeR- TOS Timer	Easy to use, does not affect the system	 Maximum resolution equals the RTOS tick cycle Timer callback functions come from the low-priority timer task. This task may be preempted by other tasks, leading to a de- crease in precision and accuracy

Comparison of Different Clocks

2.2.6 Low-Power Bluetooth Application Note

Overview

This document is a development note for ESP32 series BLE applications, mainly introducing the basic concepts of ESP32 series BLE application development, the development environment, and some generally applicable precautions in the BLE development process based on NimBLE, and organizing the necessary documents for development.

Introduction to ESP32 Series Bluetooth Features ESP32 has multiple chips that support Bluetooth, and the current support for Bluetooth can be divided into the following types:

Chip Model	Bluetooth Certifi-	Function Support
ESP32	4.2	The controller is certified by Bluetooth 4.2
ESP32-C2	5.3	The controller is certified by Bluetooth 5.3, supporting all features
		of BLE 5.0 and below
ESP32-S3, ESP32-	5.3	The controller is certified by Bluetooth 5.3, supporting all features
C3		of BLE 5.0 and below
ESP32-C6, ESP32-	6.0	Certified by Bluetooth 6.0, supporting all features of BLE 5.0 and
H2, ESP32-C61,		below, and some features above BLE 5.0
ESP32-C5		

 Table 7: Bluetooth Features of ESP32 Series Chips

For specific information about chip functions, you can choose the corresponding chip and refer to ESP32 Series Bluetooth Function Support Status

Explanation of Bluetooth Protocol Stack ESP-IDF supports two types of Bluetooth host protocol stacks:

- 1. Bluedroid: Supports both classic Bluetooth and low power Bluetooth
- 2. NimBLE: Only supports low power Bluetooth

The functions of the two protocol stacks are consistent, but each has its own advantages and disadvantages, as follows:

- 1. Bluedroid: The protocol stack architecture is clear, but it occupies a larger memory and firmware size
- 2. NimBLE: The protocol stack implementation is simpler, and the memory and firmware size occupation is smaller. Developers need to have a deeper understanding of the Bluetooth protocol stack

Both protocol stacks provide a wealth of examples and documents for developers to refer to:

- Bluedroid Example
- NimBLE Example

This document is mainly based on the NimBLE protocol stack, introducing the BLE development process and basic concepts. But the basic concepts and programming ideas are also applicable to the Bluedroid protocol stack.

Basic Concepts of BLE Programming Event-driven programming model:

In ESP32's NimBLE, the callback mechanism is the basis for the entire BLE application to respond to events. It allows the registration of some functions, and when specific events (such as connection establishment, disconnection, receiving notifications, reading and writing GATT characteristics, etc.) occur, the NimBLE stack automatically calls these functions for processing.

Basic Roles of BLE There are mainly the following roles in BLE:

- 1. Central Device The device that initiates the connection Responsible for managing connections and data exchange
- 2. Peripheral Device The device waiting to be connected Provides data and services
- 3. Broadcaster Only sends broadcast data Does not establish a connection Suitable for applications such as beacons
- 4. Observer Only receives broadcast data Does not establish a connection Suitable for data collection scenarios

In practical applications, a device can support multiple roles at the same time. For example, a smartwatch can act as a slave device connecting to a mobile phone, while also acting as a master device connecting to a heart rate band.

GAP (Generic Access Profile) Responsible for BLE device discovery, connection management, broadcasting, etc.

GATT (Generic Attribute Profile) Defines the data communication format of BLE.

For more introductions to basic BLE concepts, you can refer to Introduction to Low Power Bluetooth

BLE programming considerations:

- 1. Ensure that only lightweight logic is processed in the callback. Time-consuming logic, complex logic is handled in other tasks.
- 2. Avoid calling blocking functions in callbacks, such as vTaskDelay, etc.

Protocol stack deinitialization NimBLE protocol stack deinitialization:

- 1. Disconnect all connections, turn off broadcasting and scanning
- 2. Call nimble_port_stop(); to stop the NimBLE protocol stack
- 3. Call nimble_port_freertos_deinit(); to deinitialize FreeRTOS related resources
- 4. Call nimble_port_deinit (); to deinitialize the NimBLE port
- 5. Call esp_bt_controller_disable(); to disable the Bluetooth controller
- 6. Call esp_bt_controller_deinit(); to deinitialize the Bluetooth controller

Bluedroid protocol stack deinitialization:

- 1. Disconnect all connections, turn off broadcasting and scanning
- 2. Call esp_bluedroid_disable(); to disable the Bluedroid protocol stack
- 3. Call esp_bluedroid_deinit (); to deinitialize the Bluedroid protocol stack
- 4. Call esp_bt_controller_disable(); to disable the Bluetooth controller
- 5. Call esp_bt_controller_deinit (); to deinitialize the Bluetooth controller

Broadcast and Scan

Functions of Broadcasting Broadcasting is the foundation of BLE device discovery and connection. Broadcast packets are sent by the broadcaster (Peripheral), and the master device (Central) discovers BLE devices and can establish connections with them by scanning broadcast packets.

Function	Description
Scanning/Connection Discov-	Peripheral (slave device) allows Central (master device) to discover and connect
ery	through broadcasting
Direct Data Transfer Without	Broadcast packets can carry a small amount of data, suitable for sending noti-
Connection	fications, location information, sensor data, etc.
Signal Triggering Behavior	Such as iBeacon, Eddystone used for positioning, waking up applications, etc.
Device Identity Recognition	Broadcasting includes device name, UUID, manufacturer information, etc.,
	which is convenient for identifying device types

Table 8: BLE broadcasting functions

Before reading this section, it is recommended to read the BLE Device Discovery section first to understand the introduction of BLE 4.2 broadcasting.

Common Data Types and Their Names in Broadcasting In the BLE broadcast packet, each piece of data is composed of Length + Type + Value, and the Type field is defined according to the Bluetooth SIG specification. The following are common data types in broadcast packets.

Turne	Nome (English)	Chinaga Evalana	Common Lloco/Niotoo
туре	Name (English)	Chinese Explana-	Common Uses/Notes
(Hex)		tion	
0x01	Flags	Flag bits, indicating	Indicates whether it supports BR/EDR, whether it is in
		device capabilities	broadcast mode, etc., mandatory
0x02	Incomplete List of	Incomplete 16-bit	Some of the GATT services provided by the broadcast-
	16-bit UUIDs	service UUID list	ing device
0x03	Complete List of	Complete 16-bit	Notify all services, commonly used for general BLE pe-
	16-bit UUIDs	service UUID list	ripherals
0x06	Incomplete List of	Incomplete 128-bit	Part of the custom service' s UUID (128-bit)
	128-bit UUIDs	service UUID list	
0x07	Complete List of	Complete 128-bit	Full broadcast of custom services
	128-bit UUIDs	service UUID list	
0x08	Shortened Local	Shortened device	Save space, prioritize short names
	Name	name	
0x09	Complete Local	Complete device	Broadcast device name, visible when scanning with a
	Name	name	mobile phone
0x0A	Tx Power Level	Transmission power	Used to estimate distance, in conjunction with RSSI
		level (dBm)	
0x16	Service Data - 16-	Service data with	Broadcast some service-related data (such as tempera-
	bit UUID	16-bit UUID	ture, battery level)
0xFF	Manufacturer Spe-	Manufacturer'	Widely used in iBeacon, Eddystone, custom protocols
	cific Data	s custom data	
		segment	

Table 9: BLE broadcast data types

For more broadcast data types, please refer to Assigned Numbers.

New Broadcast Features Introduced in BLE 5.0 BLE 5.0 introduces new mechanisms such as Extended Advertising and Periodic Advertising to break through some limitations of traditional broadcasting. The broadcasting in BLE 4.2 and previous versions is known as Legacy Advertising. Compared to BLE 4.x, Extended Advertising brings significant improvements and enhancements in broadcasting, with the main goal of supporting longer distances, larger data volumes, and more flexible low-power applications. Here is a comparison of the two types of broadcasting:

Fea-	BLE 4.x	BLE 5.0
ture/Difference		
Broadcast Data Ca-	Up to 31 bytes	Up to 255 bytes (Extended Advertising)
pacity		
Number of Broad-	Fixed 3 (37,38,39)	Supports more channels (all 40 available for
cast Channels		Extended Advertising)
Broadcast Distance	Typically around 50 meters	Up to 200 meters (using Coded PHY)
Support		
Broadcast Rate	Standard 1 Mbps	Supports 2 Mbps high-speed broadcast
Broadcast Mode	Legacy Advertising	
		Added Textended Advertising
		Added Extended Advertising
		Supports Periodic Advertising
Broadcast Connec-	Can only carry connection request informa-	Broadcast data and connection requests can
tion Capability	tion in broadcast	be separated, more flexible
Multi PHY Support	Only 1 Mbps	Added 2 Mbps and Coded PHY (500/125
(Physical Layer)		kbps)
Broadcast Power	Limited	Further, faster, more power-saving (accord-
Consumption vs		ing to configuration selection)
Distance/Rate		

Table 10: BLE 4.x vs BLE 5.0 Broadcast Feature Comparison

The biggest advantage of BLE 5.0 broadcasting (Extended Advertising) is the increase in broadcast data capacity. However, some mainstream mobile phones do not fully support BLE 5.0 broadcasting, and there may be problems such as connection failure, inability to search for broadcasts, and inability to parse broadcast data in actual use. If optimal compatibility is considered, it is recommended to use traditional broadcasting.

Known compatibility issues with Extended Advertising packets include:

- IOS 18.5 devices may not be able to scan for 1M PHY Extended Advertising packets
- Some Android devices may not be able to scan for non-1M PHY Extended Advertising packets

Broadcast API Usage Instructions The NimBLE protocol stack provides two sets of broadcast APIs for BLE 4.x and BLE 5.0, each with corresponding configuration items and cannot be used simultaneously. For example, the bleprph_example example defaults to using the BLE 5.0 broadcast API. If you need to use the BLE 4.x broadcast API, in addition to turning off the example configuration item EXAMPLE_EXTENDED_ADV to make the main program call the 4.x broadcast API, you also need to turn off the broadcast configuration item BT_NIMBLE_EXT_ADV.

After the launch of BLE 5.0, traditional broadcasting is considered a subset of BLE 5.0 broadcasting, and the BLE 5.0 broadcasting API can also send traditional broadcasts. For specific implementation code, refer to the ble_multi_adv_example example.

Explanation of Broadcast Parameters For the configuration of broadcast connection parameters, refer to the ble_gap_ext_adv_params structure.

The following configuration items need special attention:

- itvl_min: Unit is 1.25 ms, the value range is 0x0006 to 0x0C80
- itvl_max: Unit is 1.25 ms, the value range is 0x0006 to 0x0C80
- channel_map: Specifies the channels used for broadcasting, 0x00 represents the default three channels, 0x01, 0x02, 0x04 represent channels 37, 38, 39 respectively, 0x07 represents all three channels 37, 38, 39
- own_addr_type: 0x00, 0x01, 0x02, 0x03 represent the broadcast address types of public address, random address, resolvable public address, and resolvable random address respectively

In addition, when configuring the broadcast type, it should be noted that non-directed connectable broadcasts must be scannable.

Scanning Scanning is the basis for discovering and connecting BLE devices. The central device discovers BLE devices and can establish connections with them by scanning broadcast packets.

For theoretical knowledge related to scanning, please refer to the Basic Concepts of Scanning section.

Instructions for Using Scanning API Similar to broadcasting, in the NimBLE protocol stack, two sets of APIs are provided to start scanning, namely ble_gap_disc and ble_gap_ext_disc.

- Use the ble_gap_disc API to start scanning. The scanning-related configuration items are configured by the ble_gap_disc_params structure. For usage, refer to the examples blecent and ble_multi_adv.
- Use the ble_gap_ext_disc API to start scanning. The scanning-related configuration items are configured by the ble_gap_ext_disc_params structure. For usage, refer to the example ble_multi_conn.

Note: Avoid printing too much data in the scanning callback, otherwise the task watchdog may be triggered if there are too many surrounding devices and the printing is not timely.

Broadcast Packet Filtering Configure disc_params.filter_duplicates = 1 to enable broadcast packet filtering. However, the specific method of broadcast packet filtering is determined by the configuration item BT_CTRL_SCAN_DUPL_TYPE.

- Scan Duplicate By Device Address (default): Filters broadcast packets from the same device address
- Scan Duplicate By Advertising Data: Filter identical broadcast packets

• Scan Duplicate By Advertising Set Handle: Filters broadcast packets from the same device with the same content

The number of scan response packets and broadcast packets received is inconsistent. The protocol stipulates that devices should not send scan requests for every consecutive advertisement packet to avoid generating excessive responses. If this function is indeed necessary, ESP32-C3 and previously released chips can enable the configuration item BT_CTRL_SCAN_BACKOFF_UPPERLIMITMAX, while ESP32-C2 and later chips can force the initiation of scan requests by setting the configuration item BT_CTRL_SCAN_BACKOFF_UPPERLIMITMAX to 1.

Connection

Connection is the basis for data transmission between BLE devices. The connection is established through the interaction between the central device (Central) and the peripheral device (Peripheral). Before reading this section, it is recommended to read the connection section to understand the related introduction of BLE connection.

GAP Layer The GAP layer is the control layer of the BLE protocol stack, responsible for managing connections, disconnections, connection parameter management, etc. In NimBLE, its API interface is defined in the $ble_gap.h$ file.

The main functions of the GAP layer include:

Function Description	Related API
Device discovery and broad-	
casting	Broadcast Control: 'ble gap adv start', 'ble gap adv stop'.
	'ble_gap_adv_active'
	Broadcast data settings: 'ble_gap_adv_set_data',
	'ble_gap_adv_rsp_set_data'
	Scan Control: 'ble_gap_disc', 'ble_gap_disc_cancel',
	Extended Broadcast: 'ble_gap_ext_adv_*' related functions
	Periodic Broadcasting: 'ble_gap_periodic_adv_*' related functions
Connection management	
	Establish connection: 'ble_gap_connect', 'ble_gap_ext_connect'
	Connection Parameter Update: 'ble_gap_update_params'
	Connection fermination: 'ble_gap_terminate Connection status query: 'ble_gap_conn_find' 'ble_gap_conn_active'
	Multi-connection support: 'ble gap multi connect'
Security related	
	Pairing and Encryption: 'ble_gap_security_initiate',
	'ble_gap_pair_initiate'
	Encryption Control: 'ble_gap_encryption_initiate'
	Dipair: Die_gap_unpair , Die_gap_unpair_oldest_peer
	Thracy Wode. bic_gap_set_priv_mode
Physical layer parameter con-	
trol	PHY Settings: 'ble_gap_set_prefered_le_phy', 'ble_gap_read_le_phy'
	Data Length Control: 'ble_gap_set_data_len'
	Subrate Control: 'ble_gap_set_default_subrate', 'ble_gap_subrate_req'
Signal quality monitoring	
Signal quality monitoring	
	RSSI Reading: ble_gap_conn_rssi
	Transmit Power Report: 'ble gap set transmit power reporting enable'
Event handling	
	Event callback registration: 'ble_gap_set_event_cb'
	Event Listener: 'ble_gap_event_listener_register'
	Supports multiple event types, such as connection, disconnection, encryption
	status change, etc.
Test function	
	DTM(Direct Test Mode): 'ble_gap_dtm_tx_start',
	'ble_gap_dtm_rx_start'
	Enhanced DTM: 'ble_gap_dtm_enh_tx_start',
	ole_gap_dtiii_eiiii_ix_start
Other functions	
	Whitelist Management: 'ble_gap_wl_set'
	Device Information Reading: 'ble_gap_read_rem_ver_info'
Espressif Systems	Address Resolution 141 ble_gap_rd_local_resolv_addr' Release master
	Submit Document Feedback

Table 11: Main fu	inctions of t	he GAP layer
-------------------	---------------	--------------
Obtain the list of paired devices In NimBLE, you can obtain the list of paired devices by calling the ble_store_util_bonded_peers() function. This function will return all the information of the bonded devices, including the device address, keys, etc.

Set Channel Map In NimBLE, the Channel Map can be set by calling the This function internally invokes the Set_Host_Chan_Class ble_hs_hci_set_chan_class() function. command, informing the controller which data channels are faulty and should be avoided in future connections. It should be noted that this setting only affects the channels available to the controller when establishing future connections or scanning/broadcasting, it does not trigger the sending of the LL_CHANNEL_MAP_REQ command, nor does it change the channel mapping of existing connections. The controller may use this information to avoid faulty channels when establishing future connections, but it will not trigger channel map updates for existing connections.

Reference examples: Bluedroid:

- GATT Server example
- GATT Client example

NimBLE:

- BLE Central example
- BLE Peripheral example

Multiple connections Multiple connections is a feature of BLE devices, where a master device can connect to multiple slave devices at the same time.

Reference examples: Bluedroid:

- GATT Multi-connection Client example
- GATT Server/Client Coexistence example

NimBLE:

• BLE Multi-connection example

BLE Security

Concepts of pairing and bonding: Pairing: Two devices agree to establish a connection with a certain level of security.

Bonding: At least one device sends security-related information (such as Long Term Key (LTK), Connection Signature Resolution Key (CSRK), or Identity Resolution Key (IRK)) to another device for future connections. After successful pairing, if bonding is supported, key distribution will be carried out; otherwise, bonding information will not be exchanged. Pairing may occur without necessarily requiring bonding. During the pairing process, devices will exchange attributes to determine whether the other party supports bonding. If neither party supports bonding, the security information of the other party will not be stored.

In NimBLE, the configuration related to pairing and bonding is defined by the ble_hs_cfg structure, and users can enable the corresponding configuration items as needed:

- sm_io_cap: Set IO capabilities, used for authentication methods during the pairing process
- sm_bonding: Whether to enable the bonding function
- sm_our_key_dist: The type of key distributed by this device
 - BLE_SM_PAIR_KEY_DIST_ENC: Encryption Key
 - BLE_SM_PAIR_KEY_DIST_ID: Identity Resolution Key (IRK)
- sm_their_key_dist: Type of key distributed by the peer device
 - BLE_SM_PAIR_KEY_DIST_ENC: Encryption Key
 - BLE_SM_PAIR_KEY_DIST_ID: Identity Resolution Key (IRK)

- sm_mitm: Whether to enable man-in-the-middle protection
- sm_sc: Whether to enable secure connections (LE Secure Connections)

Initiate pairing request actively: In NimBLE, you can actively initiate a pairing request through the ble_gap_security_initiate() function.

Initiate pairing request from the mobile end: You can also initiate a pairing request by calling the corresponding API on the mobile end. In addition, accessing encrypted services or properties will also automatically trigger pairing.

Performance Optimization

Transmit Power Both esp_ble_tx_power_set and esp_ble_tx_power_set_enhanced can be used to set the transmission power. It is recommended to use the esp_ble_tx_power_set_enhanced function, as it supports a more granular range of power. Different chips support different power ranges, which can be defined in the corresponding macros in the esp_bt.h file.

Power Consumption Optimization How to optimize power consumption:

- 1. Use low power mode
 - Use the Auto Sleep mode to allow the RF and CPU to enter low power mode when idle.
- 2. Use an external crystal oscillator
 - Using an external 32.768 kHz crystal can provide a more stable clock source.
 - Helps to improve timing accuracy in low power mode
 - Reduce the power consumption of the internal RC oscillator
- 3. Choose appropriate connection parameters
 - Connection Interval: Choose an appropriate time interval based on application requirements. A longer interval can reduce power consumption.
 - Slave Latency: Allows the slave device to skip a certain number of connection events, further reducing power consumption.
 - Supervision Timeout: Set a reasonable timeout period to avoid unnecessary reconnections.
 - Use a larger MTU size to reduce the number of data transfers

4. Other optimization suggestions

- Design packet sizes reasonably to avoid fragmented transmission.
- Use notifications (Notification) instead of indications (Indication) where possible.
- Utilize the power management features of ESP32, such as dynamic frequency adjustment
- 5. Appropriately reduce transmit power

Typical power consumption data:

Chip	Max Current	Modem Sleep	Light Sleep (Main	Light Sleep
			XTAL)	(32KHz XTAL)
ESP32	231 mA	14.1 mA	Х	1.9 mA
ESP32C3	262 mA	12 mA	2.3 mA	140 uA
ESP32S3	240 mA	17.9 mA	3.3 mA	230 uA
ESP32C6	240 mA	22 mA	3.3 mA	34 uA
ESP32H2	82 mA	16.0 mA	4.0 mA	24 uA
ESP32C2	130 mA	18.0 mA	2.5 mA	169 uA

Reference example: BLE Low Power Example

Transmission Rate Optimization BLE 5.0 and above versions support multiple physical layers: - LE 1M UN-CODED PHY: Bit transmission rate is 1M bit/s (Physical layer used by BLE 4.2) - LE 2M UNCODED PHY: Bit transmission rate is 2M bit/s - LE CODED PHY: Used for long-distance communication, bit transmission rate is 125 K bit/s or 500 K bit/s

Factors affecting the actual transmission rate:

1. Protocol layer encapsulation overhead

- Data needs to be encapsulated layer by layer from the ATT layer to the LL layer.
- A protocol header (Header) is added to each layer.
- The Preamble of LE 1M UNCODED PHY is 1 byte
- The Preamble of LE 2M UNCODED PHY is 2 bytes

2. Data Length Extension (DLE)

- BLE 4.0/4.1: Maximum ATT Payload is 20 bytes
- BLE 4.2/5.0: Supports DLE, maximum ATT Payload can reach 244 bytes
- It is recommended to configure DLE to its maximum value to improve the utilization rate of single packet data.
- 3. Inter-Frame Space (T_IFS)
 - The distance between two air packets on the same data channel is 150 us.
 - This is a fixed time consumption

4. Connection event and connection interval

- In each connection event, both parties need to complete a packet transmission and reception once.
- Even if there is no data, an empty packet still needs to be sent.
- A connection event can include multiple packet interactions
- The number of packets is related to the size of the connection interval

Optimization suggestions:

- 1. Application layer packet sending method
 - Use write command instead of write request
 - Use Notify instead of Indication

2. Connection interval optimization

- Adjust packet size to increase packet fill rate during connection events.
- Adjust the connection interval appropriately to send more data packets in a single connection event.
- 3. Use 2M PHY
 - NimBLE uses the ble_gap_set_prefered_le_phy function to set the physical layer for a specific connection, and the ble_gap_set_prefered_default_le_phy function to set the default physical layer.
 - Bluedroid uses the esp_ble_gap_set_preferred_phy function to set the physical layer of a specific connection, and uses the esp_ble_gap_set_preferred_default_phy function to set the default physical layer.

Reference examples

- NimBLE throughput example
- Bluedroid throughput example

Problem Quick Check

Information Needed for Problem Reporting When encountering a problem, please first collect the following information:

- The chip being used
- IDF version
- The protocol stack being used
- Complete logs
- The time and frequency of the problem occurrence
- The operation steps when the problem occurred

If the product is still in the development stage, it is recommended to upgrade to the latest TAG version of the current release.

NimBLE Error Codes NimBLE error codes are divided into the following categories:

-	· · · · · · · · · · · · · · · · · · ·	
Error Type	Description	Error Return Values from Different Lay-
		ers
NimBLE Core	Error codes defined by NimBLE itself	(X)
ATT	Attribute protocol layer error codes	0x100+(X)
HCI	Host Controller Interface layer error codes	0x200+(X)
L2CAP	Logical Link Control and Adaptation Layer	0x300+(X)
	Protocol error codes	
Security manager	Security management layer error codes (lo-	0x400+(X)
(us)	cal device)	
Security manager	Security management layer error codes (re-	0x500+(X)
(peer)	mote device)	
Hardware errors	Hardware layer error codes	0x600+(X)

Table	13:	NimBL	E Error	Code	Types
-------	-----	-------	---------	------	-------

Where X represents the specific error type number. For example, if 0x102 is returned, it means the ATT layer error code 0x02. If the returned error code is in decimal, it needs to be converted to hexadecimal. Then check the corresponding error type.

For specific error codes, please refer to ble_hs.h

HCI Error Codes HCI error codes are error codes returned by the Bluetooth Host Controller Interface (HCI) layer, indicating the specific reason for the operation failure. The following are common HCI error codes and their descriptions:

Error Code	English Description	Description	Possible Causes
0x00	Success	Success	Operation successfully com- pleted
0x01	Unknown HCI Command	Unknown HCI command	The controller does not under- stand the operation code of the HCI command packet sent by the host
0x02	Unknown Connection Identi- fier	Unknown connection identi- fier	The connection specified in the command does not exist or the connection type is incor- rect
0x03	Hardware Failure	Hardware failure	A failure occurred in the controller that cannot be de- scribed by other error codes
0x05	Authentication Failure	Authentication failure	Pairing or authentication pro- cess failed, possibly due to in- correct password or link key
0x08	Connection Timeout	Connection timeout	Link supervision timeout or broadcast synchronization timeout
0x0B	Connection Already Exists	Connection already exists	Attempt to create a new con- nection to a device that is al- ready connected
0x0C	Command Disallowed	Command disallowed	The controller cannot handle the command in its current state
0x0D	Connection Rejected due to Limited Resources	Limited resources	Connection rejected due to in- sufficient controller resources
			agentiques on payt page

Table 14: Common HCI Error Code

continues on next page

Error Code	English Description	Description	Possible Causes
	Connection Rejected Due To	Security reasons	Connection rejected due to
UNUL	Security Reasons	Security reasons	unmet security requirements
	Security Reasons		(such as authentication or
			(such as authentication of
0v11	Unsupported Easture or De	Unsupported feature or no	The feature or parameter
UXII	Clisupported Feature of Fa-	remeter value	the leature of parameter
	rameter value	rameter value	value in the HCI command is
0.10			not supported
0x12	Invalid HCI Command Pa-	Invalid HCI command param-	At least one HCI command
	rameters	eters	parameter is invalid
0x13	Remote User Terminated	Remote user terminated con-	The user on the remote device
	Connection	nection	terminated the connection or
			stopped broadcasting
0x16	Connection Terminated By	Local host terminated con-	The local device terminated
	Local Host	nection	the connection, synchroniza-
			tion, or broadcast
0x1E	Invalid LMP Parameters / In-	Invalid LMP parameters/LL	LMP PDU or LL control
	valid LL Parameters	parameters	PDU parameters are invalid
0x1F	Unspecified Error	Unspecified error	No other suitable error codes
0x22	LMP Response Timeout / LL	LMP response timeout/LL re-	LMP or LL transaction did
	Response Timeout	sponse timeout	not respond within the time-
			out period
0x2F	Insufficient Security	Insufficient security	The command or PDU needs
			to be executed on an en-
			crypted link
0x3B	Unacceptable Connection Pa-	Unacceptable connection pa-	The remote device terminated
	rameters	rameters	the connection due to unac-
			ceptable connection parame-
			ters
0x3C	Advertising Timeout	Advertising timeout	Fixed time broadcasting com-
		E E	pleted or directed broadcast-
			ing did not establish a connec-
			tion
0x3D	Connection Terminated due	MIC failure	Connection terminated due to
	to MIC Failure		message integrity check fail-
			ure
0x3E	Connection Failed to be Es-	Connection failed to be estab-	Connection establishment
	tablished / Synchronization	lished/Synchronization time-	failed or failed to synchronize
	Timeout	out	with periodic broadcasting
0x3F	MAC Connection Failed	MAC connection failed	MAC layer failed during the
			connection establishment pro-
			cess
0x40	Coarse Clock Adjustment Re-	Coarse clock adjustment re-	Unable to coarsely adjust the
	iected but Will Try to Adjust	iected but will try using clock	clock with the provided pa-
	Using Clock Dragging	dragoing	rameters will try using clock
			dragoing
0x41	Type() Subman Not Defined	Type () subman not defined	I MP PDI was rejected be
UNTI	Typeo Suomap Not Denned		cause the type 0 subman is
			currently not defined
0x42	Unknown Advartising Idanti	Unknown advartising identi	The broadcast or synchronize
0342	for	for	tion handle specified in the
			command does not exist
0x42	Limit Daachad	计页旧生	The number of recreated and
0345		心判限前	antions has been reached
			tivity has been reached, ac-
			uvity has been completed

Table 14 – continued from previous page

continues on next page

Error Code	English Description	Description	Possible Causes
0x44	Operation Cancelled by Host	Operation cancelled by the	The request issued by the host
		host	to the controller that has not
			yet been completed has been
			successfully cancelled
0x45	Packet Too Long	Packet is too long	The length of the packet ex-
			ceeds the maximum allowed
			length
0x46	Too Late	Too late	The operation execution time
			has passed
0x47	Too Early	Too early	The operation execution time
			has not yet arrived
0x48	Insufficient Channels	Insufficient channels	There are not enough channels
			to perform the requested op-
			eration

Table 14 - continued from previous page

For more HCI error codes, please refer to esp_bt_defs.h

BLE Packet Capture For capturing BLE broadcast packets and regular connection packets, you can refer to the following tutorial:

Windows Bluetooth Packet Capture Tutorial

The basic steps for capturing encrypted BLE connections are similar to capturing regular connections, with the main difference being the need to obtain and input the LTK (Long Term Key) of the connection. The method to obtain the LTK is as follows:

- 1. For devices using the Bluedroid protocol stack:
 - Add a print statement in the btsnd_hcic_ble_ltk_req_reply function
- 2. For devices using the NimBLE protocol stack:
 - Set the log level of NimBLE to DEBUG, then recompile the firmware

After obtaining the LTK, configure the corresponding LTK in the packet capture tool to decrypt and view the data packets of the encrypted connection.

Resource Summary

This section summarizes commonly used resources in BLE development, including official documents such as Bluetooth Core Specification, Assigned Numbers, as well as some useful tools and example codes. These resources can help developers better understand BLE technology and provide references during the development process.

Bluetooth Core Specification Bluetooth Core Specification

Assigned Numbers Assigned Numbers

ESP-IDF Bluetooth API Reference ESP-IDF Bluetooth API Reference

ESP-IDF Bluetooth API Guide ESP-IDF Bluetooth API Guide

ESP-IDF Bluetooth FAQ ESP-IDF Bluetooth FAQ

2.3 Introduction to Application Solutions

Each section within this module showcases Espressif's application solutions, often including an overview of the solution, presentation of application scenarios, and reference materials, among others.

2.3.1 AI Solution

Introduction to Edge AI Solutions

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Overview of Edge AI Solutions

Solution Features:

• Espressif provides a complete set of AIoT smart IoT solutions, fully integrating ESP-EYE development board, ESP-BOX development board, ESP-WHO AI framework, ESP-IDF software SDK, to facilitate the successful implementation of AI technology in IoT application development.

Solution Advantages:

- 1. The ESP32-S3 chip has industry-leading RF performance and integrates corresponding hardwarelevel AI acceleration, supporting the joint construction and integration of AI and IoT.
- 2. Espressif's self-developed AFE algorithm has passed Amazon Alexa's far-field test and Audio Front End certification. Working in conjunction with Espressif's self-developed WakeNet wake word engine, it aligns with Amazon's multi-language testing requirements.
- 3. Espressif's AFE algorithm is optimized based on the AI accelerator of the ESP32-S3 SoC, consuming only about 22% of CPU space, 48 KB SRAM, and 1.1 MB PSRAM, providing ample resource space for customer applications built with ESP32-S3.
- 4. ESP-Skainet works in conjunction with the WakeNet wake word engine, the offline voice command recognition engine (MultiNet), and the front-end acoustic algorithm (AFE) to provide a voice assistant with high accuracy and support for custom wake words.

Common Application Scenarios of Edge AI Espressif Voice Intelligent Conference Room

 Chinese Video Espressif Voice Intelligent Conference Room: The ESP32-S3-BOX integrates AI voice, touch screen control, sensors, infrared controllers, and smart gateways, serving as the control center for all devices in the house. It supports users to easily achieve smart home linkage through voice commands or touch screen control. Combining ESP32-S3-BOX with Espressif's AI image processing, Wi-Fi smart gateway, Wi-Fi human detection, Wi-Fi wireless image transmission and other technical solutions makes Espressif's meetings smarter.

Smart Farm

• Chinese Video ESP32 Smart Farm: The smart panel of the smart farm solution is developed by the ESP32-S3-LCD-EV-BOARD development board, integrating AI voice, touch screen control, and sensors, providing cloud connection and voice control functions. It also supports remote control from mobile applications using ESP RainMaker.

Smart Speaker

• Chinese Video ESP32 Smart Speaker: The smart speaker solution uses the ESP-BOX development board with the ESP32-S3 chip, integrating Wi-Fi, dual microphones, and a touch screen, and can be used with ChatGPT to create a smart voice assistant.

AI LLM Solution

For details on the LLM solution, see Overview of LLM Solution

AI Reference Materials (Voice)

- AI Software References
 - ESP-DL Deep Learning Framework
 - ESP ChatGPT Example
 - ESP-Skainet Voice Assistant Framework
 - ESP-SR Voice Recognition Framework
- AI Related Modules/Development Board Purchase
 - ESP32-S3 Chip/Module Purchase
 - Purchase ESP32-S3-Korvo-2
- Module/Development Board Information and Selection Reference
 - Introduction to ESP32-S3 Chip
 - Introduction to Espressif Audio Development Board
 - Espressif Product Selection Tool

AI Reference Materials (Image)

- AI Software Reference
 - ESP-DL Deep Learning Framework
 - ESP Activity Detection Example
 - ESP Gesture Detection Example
 - ESP Human Detection Example
 - ESP-WHO Image Processing Development Platform
 - ESP32-P4 Pedestrian Detection Example
 - ESP32-P4 Face Recognition Example
- AI Related Module/Development Board Purchase
 - Purchase ESP32-S3 Chip/Module
 - Purchase ESP32-S3-EYE Development Board
- Module/Development Board Information and Selection Reference
 - Introduction to ESP32-S3 Chip
 - Introduction to ESP32-S3-EYE Development Board
 - Introduction to ESP32-P4-EYE Development Board
 - ESP32-P4-Function-EV-Board Development Board Introduction
 - Espressif Product Selection Tool

Introduction to LLM Solution

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

LLM Solution Overview

Solution Features:

• The rise of large models like ChatGPT has driven a global AI boom, with cloud platforms upgrading their intelligence and AI technology continuously penetrating various industries. Espressif has built a solid technical foundation with its open, shared, and ecosystem-based intelligent hardware platform.

- Espressif provides voice and vision large model solution references, with corresponding solutions for low-cost C3, flagship S3, high-performance P4, and dual-band C5.
- Espressif is also working on large language model privatization deployment to provide strong support for accelerating customer product implementation.



Voice Solution Details

Positioning	C Series (ESP32-C3/C5)	S Series (ESP32-S3)	
Entry Level	ESP-Hi - 0.96" 160 * 80 LCD - No Codec		
	Required		
Cost-Effective	ESP-Spot - No Screen - 5G Band	ESP-Gogo & ESP-Eyemoji	
		• 1.85" 360*360 & 0.71" 160 * 160	
		LCD	
		Dual HD Screens	
High Performance		ESP-Cat - 1.85" 360*360 LCD - Dual Mi-	
		crophone Array	

Vision Solution Details

Table 16: Vision Solution Product Matrix

Positioning	S Series (ESP32-S3)	P Series (ESP32-P4)
Cost-Effective	ESP-Sparkbot - DVP VGA/720p Camera -	ESP-Brookesia - USB Camera - 720P
	1.54" 240*240 LCD	Touch Screen

Solution Overview Diagram:

Features	ESP32-S3	ESP32-C3	ESP32-C5
Recommended Sce-	Multi-modal interaction,	Cost-effective lightweight	5G advantage scenarios:
narios	voice and vision terminals	edge applications	anti-interference, network
	with display		compatibility
Voice	Multiple microphones,	Single microphone solu-	Single microphone solu-
	supports AEC echo cancel-	tion, basic collection	tion, basic collection
	lation		
Display	Multiple display solutions	Limited IO resources, re-	Slightly more IO resources
	& complex interface inter-	stricted display solutions	than C3, restricted display
	action		solutions
Camera	Supports SPI / DVP / USB	Only supports SPI camera	Only supports SPI camera
	cameras		
Touch	Built-in Touch sensor	No Touch, requires exter-	No Touch, requires exter-
		nal touch chip	nal touch chip
Memory	Up to 512KB SRAM +	400KB SRAM	Up to 384KB SRAM +
	PSRAM support		PSRAM support
Local AI Capability	Vector computing instruc-	Supports lightweight mod-	Supports lightweight mod-
	tions + neural network ac-	els	els
	celerator		

Chip Feature Comparison

Table 17: ESP32-S3 vs ESP32-C3 vs ESP32-C5 LLM Application Selection Comparison

Table 18: LLM Solution Product Matrix Overview

Category	Solution Details
Cloud Platform In-	
tegration	RainMaker & Matter: - Provides rapid cloud platform access capability
Cloud Server	
	MCP & Multi-platform Access & Offline Deployment: - Supports multiple cloud services and privatization deployment solutions
Software Frame- work	ESP-Brookesia: - Provides complete software development framework support
Application Solu- tions	 Audio & Image Solutions: Provides multiple chip and detailed scenario choices AI Empowerment: ESP-PDD: MIC/SPK/LCD small module for rapid upgrade and evaluation of existing products AT Commands: Simple and quick integration

Process Architecture Introduction: ESP chips as the edge side mainly implement data collection, preliminary processing, and transmission. Due to processing performance limitations, LLM-related processing still relies on cloud servers. Below is the overall system architecture:

Edge Tasks	Cloud Tasks
 Edge Tasks Data Collection and Preliminary Processing: Real-time collection and preliminary processing of voice and image data through Espressif chips; Local AI Model Inference: Deploy lightweight models for offline or low-latency processing; Real-time Transmission: Utilize full-duplex RTC protocol to ensure timely data transmission to the cloud. 	 Cloud Tasks Large Model Training and Deep Analysis: Perform complex calculations on collected data to provide intelligent decision support; Algorithm Updates and Optimization: Cloud computing resources used for model iteration, real-time feedback to the edge; Remote Updates and Maintenance: Implement continuous system updates and maintenance through OTA and other methods.

Table 19: Edge and Cloud Task Division

Privatization Deployment Value:

- Can accelerate testing and improve stability
- Combined with embedded devices, can achieve low-latency, high-privacy smart home experience
- Downstream enterprises can quickly integrate intelligent interaction capabilities by purchasing complete deployment solutions, lowering technical barriers and shortening product implementation cycles

Privatization Deployment Architecture:



Common LLM Application Scenarios

Application Sce-	Product Form	Recommended Solution
nario		
Plush/Desktop Pet		
Toys	 With Screen: Supports expression/interaction display Without Screen: Pure voice and action interaction 	 ESP-Eyemoji: Cute expressions, voice interaction ESP-Spot: Lightweight without screen, supports gestures
Smart Speaker		
	 Single Mic: Near-field interaction Dual Mic: Sound source localization 	 ESP-Hi: Ultra-low cost single mic solution ESP-Cat: Dual mic array, far-field wake-up
Automotive Appli- cations	 Single Screen: Driving information, cute eye expression display Dual Screen: Driving information, dual eye expression display 	 ESP-Cat: Single screen voice interaction ESP-Gogo & ESP-Eyemoji: Dual screen voice interaction
AI Empower- ment for Existing Products	SPK/MIC/LCD Small Module	 ESP-PDD: Rapid AI product formation/evaluation AT Commands: Simple and quick integration

ESP-Spot

• ESP-Spot: ESP-Spot is an AI action voice interaction core module based on ESP32-S3 / ESP32-C5, focusing on voice interaction, AI perception, and intelligent control. It not only has offline voice wake-up and AI dialogue functions but also can achieve doll touch perception through ESP32-S3' s built-in touch/proximity sensing peripherals. The device has a built-in accelerometer that can recognize postures and actions, enabling richer interactions.

Related Links:

- JLCPCB: https://oshwhub.com/esp-college/esp-spot
- Related Videos: https://www.bilibili.com/video/BV1ekRAYVEZ1/?share_source=copy_web&vd_source= 819d03f30389b111c508986ee27e0332

Features:

- Screenless solution, focusing on voice and action interaction
- Low cost, no display screen
- Can expand panel to dual-screen ESP-Gogo and ESP-Eyemoji
- S3/C5 dual adaptation, can push C5 5GHz

ESP-SparkBot

• ESP-SparkBot: ESP-SparkBot is based on ESP32-S3, integrating voice interaction, image recognition, and multimedia entertainment. It can transform into a remote control car, play with local AI, support large model dialogue, real-time video transmission, and HD video projection. Powerful performance, endless fun!

Related Links:

- JLCPCB: https://oshwhub.com/esp-college/esp-sparkbot
- Related Videos: ESP32 Large Model AI Desktop Robot ESP32 Integration with Doubao Large Model [RTC Continuous Dialogue] DeepSeek Voice Dialogue Robot: Experience High-Quality AI Dialogue with ESP32

ESP-Hi

• ESP-Hi: ESP-Hi is a high-integration AI voice solution based on ESP32-C3, using ESP32-C3' s built-in ADC as the microphone collection device and I2S PDM directly as audio output, achieving low board-level material cost.

Description:

- High Integration: Uses ESP32-C3's built-in ADC as the microphone collection device. Uses I2S PDM directly as audio output, thus eliminating the need for external CODEC chip. Achieves low board-level material cost.
- Low Resource Usage: Audio transceiver only uses 4 IO ports, uses very little CPU and memory, reserves sufficient resources for application development.
- Multiple Interaction Methods: With screen and LED indicators, supports buttons, shaking, and voice wake-up.

Related Links:

- Code Repository: Updating
- Related Videos: https://www.bilibili.com/video/BV1BHJtz6E2S/

Features:

- Currently the lowest board-level material cost AI voice solution
- C3 wake word lightweight model, supports offline wake-up

ESP-P4 Phone

• ESP-P4 Phone: A handheld device solution with screen based on ESP32-P4, combined with ESP-Brookesia' s Phone UI functionality, achieving Android-like system effects.

Hardware:

- Main Control: ESP32-P4
- Wi-Fi: ESP32-C6
- LCD & Touch: 720P MIPI-DSI ILI9881 & GT911
- Audio: 8311
- Type-C: USB2.0

Related Links:

- Code Repository: https://gitlab.espressif.cn:6688/ae_group/tools/esp-dev-tools/-/tree/feat/embedded_ world_demo?ref_type=heads
- Related Videos: To be added

Features:

- 720P high-resolution touch screen
- "ESP32-C6 + ESP-Hosted" Wi-Fi solution
- Android-like system effect, provides common functionality (such as network configuration) App

ESP-Cat

• ESP-Cat: Doubao customized version AI speaker, supports dual-mic sound source localization rechargeable round screen device, can rotate direction with turntable, has touch and battery functions.

Hardware:

• ESP32-S3

Related Links:

• Related Videos: Updating

LLM Hardware Solution Summary

Solu- tion No.	Solution Type	Resource Usage	Cost	Effects and Recommended Appli- cation Scenarios
1	Digital Microphone (MSM261S4030H0R etc.)	1 I2S (3 pins)	High	 Simple wiring Cannot achieve echo cancellation Suitable for DIY / board area limited scenarios
2	Dedicated Audio ADC + Analog Microphone (ES7210)	1 I2S + 1 I2C (5 pins)	Medium High	Recommended for multiple mi- crophone scenarios
3	CODEC + Analog Microphone (ES8311 etc.)	1 I2S + 1 I2C (5 pins)	Medium	 Low cost Good effect Recommended for use
4	Internal ADC + Op- amp + Analog Micro- phone	1 Internal ADC (1 pin)	Lowest	 Lowest cost audio input solution Meets basic audio input requirements

Audio Input Solution Comparison Table

Audio Output Solution Comparison Table

Solu- tion No.	Solution Type	Resource Usage	Cost	Effects and Recommended Appli- cation Scenarios
1	I2S Digital Amplifier (MAX98357A etc.)	1 I2S + 1 PA control (4 pins)	Medium	Good effect but no volume controlSuitable for products only requiring audio output
2	CODEC + Analog Ampli- fier (ES8311 + NS4150)	1 I2S + 1 I2C + 1 PA control (6 pins)	Low	 Optimal cost and effect Recommended for AI audio
3	I2S PDM + Analog Ampli- fier (NS4150)	2 I2S pins + 1 PA con- trol (3 pins)	Low- est	 Lowest cost but uses CPU resources Suitable for cost-sensitive scenarios

The above audio solutions can be freely combined, but if customers are in the design phase, considering cost and performance, only the following solutions are recommended:

Catagori		Colution Footures	Decembranded Lland	Deference Develop
Category	scription	Solution Features	ware	ment Board
Optimal Cost	Analog Mic + OPA Internal ADC Audio Collection + I2S PDM Output	 Single micro- phone input, mono playback Implements basic audio collection and playback Minimum IO us- age Suitable for close-range dia- logue, low-cost applications 	ESP32-C3 / ESP32-C5	ESP-HI
Balanced Choice	Single Mi- crophone + External Codec Chip (e.g., ES8311)	 Supports echo cancellation Good single mi- crophone input effect Balanced perfor- mance and cost 	ESP32-S3 / ESP32-P4 / ESP32-C5	ESP32-P4-Function- EV-Board
Best Per- formance	Multiple Mi- crophones + External Decoder + External Audio ADC Chip (e.g., ES8311 + ES7210)	 Supports far-field voice wake-up Supports echo cancellation, noise suppression Suitable for high-performance voice applications 	ESP32-S3 / ESP32-P4	ESP32-S3-Korvo-2

Voice Solution Recommendation Comparison Table

AI Vision Hardware Solution Comparison Table

Inter- face Type	Camera Per- formance	Supported Chips	Reference Development Board	Features
SPI	Low Resolution Image (max 320×240)	ESP32-S3 / ESP32-C5 / ESP32-C3	ESP32-S3- EYE	 Simple interface, suitable for beginners Lowest cost Limited frame rate and resolution Mainly used for image recognition, simple detection
DVP (Par- allel)	Low to Medium Resolution (e.g., VGA, 720p)	ESP32-S3 / ESP32-P4	ESP32-S3- EYE	 Older but mature interface Can support basic video streaming High resource usage (requires many GPIOs) Medium imaging quality, suitable for medium visual tasks
USB	Low to High Resolution (depends on USB camera)	ESP32-S3 / ESP32-P4	ESP32-S3- USB-OTG	 Plug and play with UVC cameras Supports high resolution, high frame rate Complex software support (requires USB Host capability) Suitable for applications requiring high image quality
MIPI CSI	High Reso- lution Image (e.g., 1080p, 4K)	ESP32-S3 / ESP32-P4	ESP32-P4- Function-EV- Board	 High bandwidth, low power consumption High hardware requirements (requires MIPI PHY) Supports HD, low-latency image capture Suitable for edge AI, visual analysis, recognition scenarios

In summary: ESP32-C3 / ESP32-C5 only support SPI interface cameras, currently adapted to BD3901 camera. For ESP32-S3 with clarity requirements, DVP camera is first recommended, for ESP32-P4, MIPI camera is first recommended.

Edge AI Information Sharing

- 1. Voice Capability Enhancement
 - Wake word support related repository references:
 - HIESP

– nihaoxiaozhi

- TTS supports low-cost custom wake words (expected to be released by end of May)
- 2. Vision Capability Enhancement
 - Supports local YOLO model operation
 - Can implement basic object detection functionality
- 3. Large Model Integration Examples
 - Doubao Large Model LLM Solution
 - Baidu DuerOS 3.0 Cloud
 - Speech Recognition and Speech Synthesis Library
 - Corresponding Documentation

2.3.2 Brushless Motor Solution

Introduction to Brushless Motor Solution

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Overview and Advantages of BLDC Solution Espressif's Brushless Motor (BLDC) solution offers excellent performance and stability, and can be paired with ESP master chips that support MCPWM peripherals. This solution performs well in various application scenarios such as household appliances, industrial automation, power tools, and medical equipment. Advantages include rotor initial phase detection, sensorless speed control, etc. In addition, it supports back-EMF detection with hardware comparators and internal ADCs, enabling speed closed-loop control. Here is an overview and advantages of the BLDC solution:

- **Higher operating efficiency and performance:** Utilizing the high-performance processing capability of the ESP chip, it can monitor and estimate the operating status of the brushless motor in real time, achieving stable closed-loop control.
- Rich software and hardware references: Provides comprehensive BLDC software and hardware development materials, including detailed development documents and examples. In addition, related hardware design files are open source, making it easy for customers to quickly build prototypes.
- **Complete control scheme:** Supports sensorless square wave control based on comparators or ADCs, and sensor-based FOC. Among them, the sensor-based FOC control scheme based on SimpleFOC supports IDF framework development, meeting the needs of different user groups.

In addition, Espressif also supports the following features to further expand the application scenarios of the BLDC solution:

- **RainMaker private cloud deployment:** Integrated one-stop private cloud deployment, providing users with more convenient BLDC interaction methods and remote OTA upgrade services.
- **High-performance floating-point operation performance:** Optimizes the trigonometric and floating-point operations of the SimpleFOC algorithm, improving the algorithm efficiency of BLDC.

Common Application Scenarios of BLDC Espressif's BLDC solution is widely used in various fields, including but not limited to:

• **DC frequency conversion floor fan solution:** Integrated with Wi-Fi, BLE, RainMaker, supports speed adjustment, oscillation control, natural wind speed and other functions. Typical application scenarios are small household appliances.



• **SimpleFOC single motor drive solution:** Integrated with SimpleFOC algorithm, supports motor parameter verification, speed closed-loop, angle closed-loop and torque closed-loop. Typical application scenarios are small household appliances and industrial automation applications.



• FOC knob screen solution: Integrated with SimpleFOC algorithm, supports knob screen display, ratchet hand feel simulation, etc. Typical application scenarios are knob screen applications in small household appliances.



• Sensorless square wave ADC solution: Supports pulse method rotor initial phase detection, ADC sampling zero-crossing detection, stall protection, etc. Suitable for small household appliances and industrial automation

applications.



The summary is as follows:

So- lu- tion Cat- e- gory	Solution Name	Master Control	Motor	Function	Com- po- nent	Application Scenario
Floor Fan Solu- tion	Smart DC Frequency Conversion Floor Fan	ESP32- S3	Brushless DC Motor	Supports RainMaker remote control, speed adjustment, os- cillating control, natural wind speed, and other functions	Sen- sor- less Square Wave BLDC Com- po- nent	Smart home appliances and industrial automation scenarios
FOC	SimpleFOC	ESP32-	Brushless DC	Supports speed closed-loop,	ESP_S	IN That FO Ome
Sin-	Single Mo-	C3	Motor	angle closed-loop, and torque	Com-	appliances
gle	tor Drive			closed-loop	po-	and industrial
Mo-	Solution				nent	automation
tor						scenarios
Drive						
Solu-						
tion						
FOC	FOC Smart	ESP32-	Brushless DC	Supports boundary-	ESP_S	INCROEF Softeen
Knob	Knob Screen	S3	Motor	free/boundary-based ratchet	Com-	application
Screen				knob tactile simulation	po-	scenarios in
Solu-					nent	small home
tion						appliances

Table 21:	Overview	of BLDC Solution
14010 21.	0.01.10.0	of DED C bollation

BLDC Reference Materials In addition, we currently have some public brushless motor (BLDC) github software libraries, videos, and module materials, as follows:

- Software References
 - ESP-IOT-SOLUTION ESP_SIMPLEFOC Component
 - ESP-IOT-SOLUTION Sensorless Square Wave BLDC Component
 - ESP-IOT-SOLUTION Smart DC Variable Frequency Floor Fan Example
 - ESP-IOT-SOLUTION Smart Knob Screen Example
- Module/Development Board Materials and Option References
 - Espressif Product Selection Tool

BLDC Drive Solution Summary In the brushless motor (BLDC) solution, driving the three-phase inverter circuit is an important link. In ESP chips, the LEDC and MCPWM modules are supported for driving the three-phase inverter circuit.



MCPWM Drive Solution MCPWM is a multifunctional PWM module that supports multi-channel PWM output, external pulse width calculation, etc. Some ESP chips do not support MCPWM, such as ESP32-S2, ESP32-C3, etc. In the MCPWM drive scheme, the control of the BLDC motor is realized by configuring the phase, duty cycle, frequency, etc. of the MCPWM module.

Note:

- 1. Does it support outputting complementary waveforms to control MOS drivers?
 - Yes, the MCPWM generator module supports generating independent or complementary PWM waveforms, which can control MOS drivers by configuring output pins.

LEDC Drive Scheme LEDC is mainly used to control LEDs, but it can also generate PWM signals for driving BLDC motors. All ESP chips support PWM output driven by hardware LEDC.

Note:

What are the scenarios for using LEDC drive? Is the sensorless BLDC square wave control scheme supported?
 The ESP_SIMPLEFOC component supports configuring LEDC channels to output PWM signals for driving BLDC motors. In the scenario of sensorless square wave control scheme, considering the accuracy of the sampling moment, it is recommended to use the MCPWM drive scheme.

Summary of Common BLDC Questions Q: How to control the demand for Espressif chip selection?

• A: Currently, Espressif mainly promotes the BLDC control scheme based on ESP32-S3, which supports sensorless BLDC control schemes based on ADC or comparator and sensor FOC control schemes. In addition, users also need to consider the detailed parameters required by the application, such as the number of IOs, memory and flash size, etc. You can refer to Espressif' s official selection tool or consult business.

Q: Which version of ESP-IDF should be used for the development of BLDC control schemes?

• A: ESP-IDF 5.0 and above versions need to be ensured.

Q: Can you assist in the development of BLDC brushless motors?

• A: Yes. First, users need to determine the motor model according to application requirements, including whether it is a brushless DC motor, motor parameters, etc. At the same time, users need to clarify control requirements, including speed control, angle control, torque control, etc. Then, according to the motor type and control requirements, assist users in brushless motor development.

Q: What is a BLDC driver? How do ESP series chips cooperate with it?

• A: The drive circuit of the brushless motor (BLDC) mainly uses a three-phase inverter circuit to generate magnetic fields in different directions to drive the motor. Among them, the ESP chip mainly controls the gate drive chip to control the BLDC brushless motor by generating PWM signals through the MCPWM or LEDC module.

Q: Why is the MCPWM drive scheme recommended in the BLDC scheme?

• A: MCPWM is a dedicated peripheral in ESP chips for motor and power control. Each MCPWM peripheral includes a clock divider, three PWM timers, three PWM operators, and a capture module. By configuring the MCPWM module, accurate sampling of the BLDC brushless motor can be achieved.

Q: What is the difference between the sensorless square wave BLDC component and the ESP_SIMPLEFOC component?

Answer: The Sensorless Square Wave BLDC Component is mainly used for implementing sensorless BLDC motor control, supporting external comparators and ADC to monitor back EMF for sensorless speed closed-loop control. The ESP_SIMPLEFOC Component is a sensor-based FOC control solution based on the SimpleFOC algorithm, supporting IDF framework development, and meeting the needs of different user groups. If there are no external sensors, it is recommended to use the Sensorless Square Wave BLDC Component.

2.3.3 Camera Solution

DVP & MIPI-CSI Camera Solution Introduction

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Common Application Scenarios

- 1. Existing video demos
- ESP32-S3 Blazing Milliseconds [High Frame Rate 125 fps]: ESP32-S3 with global shutter camera, achieving high frame rate of 125 FPS. It can capture the motion trajectory of moving objects, such as high frame rate image capture for writing pens and other scenarios. Combined with ESP32-S3' s network capabilities, it' s the first choice for IoT smart camera solutions.
- ESP32-S3 Watching Cute Cats Eating [High Resolution]: ESP32-S3-EYE AI development board with a 2MP camera. High-resolution real-time viewing of cats eating, first choice for pet monitoring.
- 2. QR code scanning function
- Product barcode scanning, train ticket scanning, tax invoice scanning, resource scanning and downloading, etc.
- Auto focus + Internet
- 3. Wi-Fi, LCD "Polaroid"
- Photos taken are immediately refreshed on webpages or displayed on LCD
- 4. Scanner, smart car auxiliary navigation
- 5. Face recognition
- Technology or system for identity confirmation or identity search through facial recognition
- 6. Video surveillance, light-sensitive ambient lighting
- 7. AI applications
- Gesture recognition
- Number recognition

- Food recognition
- Posture detection

Reference Materials

- Software references
 - General camera examples
 - Camera driver library: You can refer to esp32-camera, which is Espressif' s camera driver library providing camera peripheral drivers for ESP32/ESP32-S2/ESP32-S3, including the underlying 8-bit DVP interface driver, sensor driver, and image encoding/decoding. ESP32-P4, in addition to supporting the DVP interface, also supports the MIPI CSI interface. This chip uses a new code architecture, which you can refer to in esp-video-components.
 - Face recognition solution library
 - ESP-Video camera framework components
 - Audio and video intercom solution example
 - ESP32-P4 camera test examples
- Related modules/development boards purchase
 - ESP32-S3 chip/module purchase
 - ESP32-S3-EYE development board purchase
- Module/development board materials and selection reference
 - ESP32-S3 chip introduction
 - ESP32-S3-EYE development board introduction
 - ESP32-P4-EYE development board introduction
 - ESP32-P4-Function-EV-Board development board introduction
 - Espressif product selection tool

Performance Test Data Appendix Camera solutions primarily focus on the following performance indicators:

- 1. Sensor initialization time (especially for projects that need to capture immediately after startup)
- 2. Supported resolutions and data formats
- 3. Frame rate under specified resolution and data format
- 4. Transmission rate when used with Wi-Fi
- 5. Supported image processing functions (currently mainly JPEG encoding/decoding performed through software)

Related test code can be found in the test examples.

Sensor Initialization Time

Table 22: Sensor init time

Sensor	Init time (ms)
OV3660, JPEG	604
OV3660, RGB565	1301
OV2640, JPEG	200
OV5640, JPEG	240

Note: When conducting the above tests, please note:

- A picture needs to be captured at the end of the test to verify correctness
- You can refer to More optimization methods to optimize the system configuration as much as possible.

Supported Resolutions and Data Formats The supported resolution size depends entirely on the camera's maximum performance, but the capabilities of DVP itself and CPU DMA are limited, and excessive resolution will put pressure on data transmission:

- If the DVP camera sensor can output JPEG type images, the recommended maximum resolution should not exceed 5MP
- If the DVP camera sensor cannot output JPEG type images (when the image type is YUV422/RGB565, etc.), the recommended maximum resolution should not exceed 1MP

Data formats mainly depend on the output formats supported by the camera. These primarily include:

- 1. RGB
- 2. YUV
- 3. JPEG
- 4. RAW Data
- 5. BMP
- 6. Only Y/Grayscale

When the camera itself does not support JPEG data output, ESP32 can perform JPEG compression to output JPEG data.

Note: Specifically, when requiring resolutions exceeding 1024*720, it's best to consider cameras that support JPEG encoding. It should be noted that JPEG encoding/decoding performed by ESP32 will put pressure on CPU and memory.

For the ESP32P4' s MIPI CSI interface, it supports data in RGB, YUV, and RAW Data formats.

Frame Rate Under Specified Resolution and Data Format

Table 23: Sensor init time							
Output format	Chip model	Sensor model	Resolution	Frame rate			
YUV422RGB565	ESP32-S3	SC030IOT(DVP)	640*480	30 fps			
JPEG	ESP32S3	OV5640(DVP)	1600*1200	25 fps			
Only Y/MONO	ESP32-S3/ESP32-	SC031GS(DVP)	240*240	125 fps			
	S2						
RGB RAW	ESP32-P4	SC2336(MIPI)	1920*1080	30 fps			
RGB565	ESP32-P4	OV5645(MIPI)	2592*1944	15 fps			
YUV422	ESP32-P4	SC101IOT(DVP)	1280*720	24 fps			

The rates of different cameras vary greatly. When testing JPEG rates, you should specify the JPEG compression parameters and try to shoot colorful pictures. Shooting monochrome objects will contain more low-frequency information, resulting in smaller data volume after JPEG compression, and the generated data will not be representative.

For esp32-camera, the frame rate of the same camera varies greatly when settings for data format, resolution, main clock XCLK size, and idle interval time parameters differ. To ensure frame rate, fb_count should not be less than 2 when initializing the camera. Currently, driver parameters for most cameras are not optimal, and configuration methods for different cameras are not standardized, so there is significant room for optimization of the above performance data. ESP32-S3 has an independent CAM DVP interface with a peripheral interface rate 2-3 times higher than ESP32.

Transmission Rate When Used With Wi-Fi Usually, once the user' s required data format, resolution, frame rate and other parameters are determined, the feasibility of the solution can be initially estimated based on Wi-Fi test data.

Taking JPEG@480*320@20fps as an example, a single JPEG@480*320 image is typically 30KB-50KB. With a required frame rate of 20fps, the required Wi-Fi speed should be 600KB/s-1000KB/s. By checking the ESP32-S3 Wi-Fi throughput, it can be seen that ESP32-S3 meets these requirements.

Current testing of the esp-rtsp example running on ESP32-S3 shows that 720p + MJPEG video stream can achieve a frame rate of around 20fps.

分辨率	输入图片格式	输出图片格式	转换	转换速度
QVGA	YUV422	JPEG	压缩	32.70 fps
VGA	YUV422	JPEG	压缩	8.45 fps
720P	YUV422	JPEG	压缩	2.88 fps
QVGA	YUV420	JPEG	压缩	45.05 fps
VGA	YUV420	JPEG	压缩	11.74 fps
720P	YUV420	JPEG	压缩	3.98 fps
QVGA	JPEG	RGB565	解压缩	59.42 fps
VGA	JPEG	RGB565	解压缩	15.29 fps
720P	JPEG	RGB565	解压缩	5.09 fps

ESP32-S3 Encoding/Decoding Performance ESP32-S3 does not have hardware encoding/decoding capabilities; its driver code includes the TinyJPEG software encoding component.

Common Q&A

Querying resolution and output format descriptions in esp-video. esp-video provides detailed descriptions of camera sensor output formats. Taking OV5645 as an example, users can refer to this description to understand its supported output formats and select the desired output format in menuconfig. If you' re using ESP32-P4 and encounter unsupported resolutions, you can use the PPA module to scale or crop the original output image.

Modifying ESP32-P4 ISP parameters. For sensors outputting RAW format data, the ISP module is needed to optimize image brightness and color. The ISP system consists of three parts: ISP calibration tools, ISP control algorithms, and ISP hardware pipelines. The ISP calibration tools require specialized laboratories and professional personnel to use. Some parameters of the ISP control algorithms and ISP module can be configured through JSON files. Taking OV2710 as an example, its JSON file is located here. Users can copy this default JSON file to create custom JSON files and specify the path to this file through the configuration menu.

How to determine if a sensor requires enabling the ISP Pipelines Controller? This requires checking the sensor's datasheet. Based on output characteristics, sensors can be divided into three types:

- 1. JPEG sensor: Sensors that can directly output JPEG data. The internal structure of these sensors can be simplified as: RAW sensor + Internal ISP + JPEG encoder. The Internal ISP encodes RAW data into YUV or RGB format data; the JPEG encoder encodes RGB or YUV format data into JPEG. It actually has three modules working.
- YUV sensor: Sensors that can directly output YUV422 or RGB565 data. The internal structure of these sensors can be simplified as: RAW sensor + Internal ISP. It actually has two modules working. When JPEG data is needed, the SOC needs to perform JPEG encoding. For ESP32-S3, the esp_new_jpeg component can be used for software encoding. For ESP32-P4, it has a HW_JPEG_Encoder.
- 3. RAW sensor: Sensors that can only output RAW8 or RAW10 data. The internal structure of these sensors only has a RAW sensor. Therefore, when projects need RGB, YUV, or JPEG format data, the SOC needs to deploy ISP and JPEG encoder. For projects with low efficiency requirements, software-implemented ISP can be used; otherwise, it's necessary to enable the ISP Pipelines Controller on the SOC.

For RAW sensors, how to quickly obtain JPEG images or H.264 data? First, enable the ISP Pipelines Controller on the SOC to obtain YUV or RGB format data in the ISP output queue. Then use the M2M (Memory-ToMemory) middleware provided by esp-video to transfer data from the ISP output queue to the encoder input queue. Finally, retrieve the encoded data from the encoder output queue. Examples using the M2M mechanism include: image_storage and uvc.

What to do if the image frame rate is slow? The image data users care about is often processed and transmitted through multiple modules. Therefore, it's necessary to guide customers to test the rate at each stage step by step.

First, confirm the actual output frame rate of the camera sensor. For users using esp32-camera, use the test_framerate example; for users using esp-video, use the capture_stream example. Then gradually enable ISP and encoder, and test the rate after enabling these encoding modules respectively. Finally, test the data transmission rate (network, peripheral interface).

Introduction to USB Camera Solution

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Common Application Scenarios

USB Host

• ESP32-S3 Host UVC LCD Display Solution: Espressif' s USB audio and video transmission solution implements the standard USB UVC host class, using the native USB interface to connect to a universal camera, and simultaneously achieving USB camera data stream reading, quick start, hot plugging, automatic descriptor parsing, MJPEG video stream transmission, bulk and synchronous two transmission modes, adaptive image size display, switching image transmission size, Wi-Fi image transmission and other functions on a single SoC. It can be used in scenarios such as peepholes, smart doorbells and locks, electronic endoscopes, etc.

USB Device

• ESP32-S2/S3 Device USB UVC Solution: Transmits the camera' s image to the PC host through the USB interface, can be used as a USB camera, and can also support microphone output, suitable for peepholes. - Example code: usb camera

Reference Materials

- Document References
 - USB Stream Component
 - USB Device UVC
- Software References
 - USB Stream User Guide
 - USB Stream Example
 - ESP-IDF USB Example
 - USB Example in ESP-IOT-SOLUTION
- Related Modules/Development Boards Purchase
 - ESP32-S3 Chip/Module Purchase
 - ESP32-S3-USB-OTG Purchase
- Module/Development Board Information and Selection Reference
 - ESP32-S3-USB-OTG Development Board Introduction
 - Introduction to ESP32-S3 Chip

- Espressif Product Selection Tool

USB UVC Performance Test Data Appendix

- The camera must be compatible with USB1.1 Fullspeed mode
- The camera needs to come with MJPEG compression
- In synchronous transmission mode, the total bandwidth of USB transmission for image data stream should be less than 4 Mbps (500 KB/s)
- In bulk transmission mode, the total bandwidth of USB transmission for image data stream should be less than 8.8 Mbps (1100 KB/s)
- MJPEG format, 800 x 480 @15fps under bulk transmission mode, 480 x 320 @15fps under synchronous transmission mode

2.3.4 LCD Solution

Introduction to LCD Solution

This section can directly refer to the LCD Application Solution Introduction in the ESP-IoT-Solution document.

In addition, the ESP-IoT-Solution document also provides an LCD development guide and detailed explanations of various LCD interfaces. For specific content, please refer to LCD Display.

2.3.5 ESP-CSI Solution

Introduction to ESP-CSI Solution

Features of the Solution ESP-CSI (Channel State Information) is a technology provided by Espressif for obtaining the state information of Wi-Fi channels. It utilizes the features of the ESP series chips, and by analyzing detailed indicators of Wi-Fi channels, such as phase, propagation delay, signal-to-noise ratio, channel matrix, data rate, etc., it provides more detailed and meticulous wireless channel information. This strongly supports a variety of advanced applications and intelligent sensing technologies, bringing more innovation and application possibilities.

ESP-CSI provides more information than traditional signal strength (RSSI). RSSI simply measures the strength of the signal, while CSI provides more detailed and richer channel information, including the amplitude and phase of the signal. This information is very useful for some advanced applications, such as indoor positioning, environmental monitoring, and human perception.

ESP-CSI has a wide range of application prospects in the fields of IoT, smart homes, smart cities, etc., making devices more intelligent and accurately perceive and respond to changes in the surrounding environment.

Advantages of the Solution

- No additional hardware cost: The ESP-CSI solution is based on the inherent features of the ESP series chips, without the need for additional hardware costs, providing developers with an economical and efficient solution.
- Real-time response: ESP series chips can process and analyze CSI data in real time on the device locally, providing fast response and high performance.

- Interference resistance: The amplitude of CSI is essentially a set of channel attenuation coefficients. As long as the channel itself does not change, it is very robust against interference from power adapters and other frequency hoppers.
- Finer granularity: Instead of using composite values (such as RSSI) to measure channels, it samples frequency responses in units of subcarriers in the channel, thus describing the channel in a finer granularity within the frequency domain.

Common Application Scenarios of ESP-CSI

- Intruder Detection: With ESP-CSI, you can choose a combination of highly sensitive subcarriers and signals from non-line-of-sight propagation path directions to improve the sensitivity of passive personnel detection and expand the detection range under different multipath propagation environments. This can form an intrusion detection system, providing an efficient intrusion detection solution for security applications.
- Positioning and Ranging: ESP-CSI can draw on the RSSI method and use CSI as richer fingerprint information (including signal amplitude and phase information on multiple subcarriers), or through the frequency-selective attenuation model for more accurate positioning and ranging. This helps to achieve more accurate positioning and distance measurement in indoor or complex environments.
- Human Activity Detection and Recognition: With the high sensitivity of ESP-CSI, it can recognize human movements, gestures, and daily activities. This is very useful for implementing smart home, health monitoring, and human interaction applications.
- Espressif ESP-CSI Intelligent Human Perception Detection Solution

Chip CSI Performance Ranking

• ESP32-C5 > ESP32-C6 > ESP32-C3 \approx ESP32-S3 > ESP32

Implementation of ESP-CSI



Router

Using only one ESP chip and router

- The ESP chip sends the constructed Ping (Packet Internet Groper) packets to the router. Ping is a protocol used to test the network connection with the target device, usually used to detect network latency and connection status. In Ping packets, it usually includes information such as the source IP address, target IP address, timestamp, etc.
- The router will reply with Ping packets carrying some CSI information.
- After the ESP chip captures the returned data packets, it parses these packets to extract CSI information. Parsing CSI information involves analyzing the structure and fields of the packets to obtain various parameters of channel status information, such as amplitude, phase, etc.



Exchange of CSI information between two or more ESP chips

- Two ESP chips construct their own Ping request packets respectively. The Ping request packets contain information such as the source IP address, target IP address, timestamp, etc. These Ping request packets will be sent to the router. They do not communicate directly with each other, but realize indirect communication through the router as a relay.
- After the router receives the Ping request packets sent by the two ESP chips, it determines which device the packets should be forwarded to based on the target IP address.
- After ESP chip B receives the Ping request from ESP chip A, it generates a Ping reply packet based on the information in the request, and carries the required CSI information. When generating the reply, ESP chip B records the sending timestamp for subsequent round-trip time (RTT) calculation.
- ESP chip B sends the generated Ping reply packet back to the router through the Wi-Fi module. Then the router forwards the Ping reply packet sent by ESP chip B to ESP chip A.
- After ESP chip A captures the returned packets, it parses these packets to extract CSI information. Parsing CSI information involves analyzing the structure and fields of the packets to obtain various parameters of channel status information, such as amplitude, phase, etc.



- The packet transmission device continuously sends broadcast packets on different wireless channels. Each time it sends a broadcast packet, it switches to a new wireless channel, thus covering the entire spectrum bandwidth.
- All ESP devices will receive the broadcast packets sent by the packet transmission device. Since they are all working in monitoring mode, they can simultaneously capture the broadcast packets from the packet transmission device.
- Each ESP device needs to parse these packets and extract the required CSI information after receiving the broadcast packets. Parsing broadcast packets involves analyzing the structure and fields of the packets to obtain various parameters of channel status information, such as amplitude, phase, etc.

Note: CSI data can only be obtained when the Wi-Fi packet rate is 802.11 a/g/n.

ESP-CSI References

- ESP-CSI Github
- ESP-IDF Wi-Fi Library

Common Crystal Multi-Antenna Solution The CSI research network cards used in the current algorithm deployment scenarios are mostly multi-antenna network cards. We design the common crystal of two chips, which can achieve a similar effect. esp-crab provides a Wifi-CSI RF phase synchronization solution, which includes two working modes: 1. Self-transmission and self-reception mode; 2. Single-transmission and dual-reception mode.

Common Crystal Solution Schematic and Example

• Self-transmission and reception mode: In this mode, two ESP32-C5 chips send and receive signals respectively. By calculating the phase information in the received Wi-Fi CSI signal, it is possible to sense disturbances in the radio frequency signal path at the millimeter level. At the same time, by installing copper sheets to control the transmission path of the radio frequency signal, the sensing range can also be adjusted, thus providing technical support for high-precision short-range Wi-Fi sensing. This mode makes Wi-Fi signal sensing more refined, suitable for precise sensing applications in short distances and complex environments.

Self-transmission and reception amplitude effect Self-transmission and reception phase effect

• Single-transmission dual-reception mode: In this mode, one ESP32-C5 chip is responsible for sending signals, while the two ESP32-C5 chips of esp-crab are responsible for receiving signals. By dispersing the sender and receiver, Wi-Fi sensing can be achieved in a large spatial range. The co-crystal Wi-Fi CSI information obtained by esp-crab can meet the performance requirements of Wi-Fi sensing in cutting-edge research, and can directly dock with mature advanced algorithms, further enhancing the accuracy and application value of the wireless sensing system. This mode provides strong technical support for wireless sensing and positioning in large ranges and complex environments.

Single-transmission dual-reception phase effect

Motion Detection Algorithm We have designed a multi-antenna co-crystal development board for collecting and real-time processing of CSI data. The board is equipped with an ESP32-C3 and an ESP32-S3. The ESP32-C3 is connected to three onboard directional antennas through an antenna switch, while the ESP32-S3 is connected to an antenna socket to provide more flexible antenna types and signal transmission path choices. During operation, the ESP32-C3 continuously sends airborne packets (such as ESP-NOW), while the ESP32-S3 is responsible for obtaining CSI from the airborne packets sent by the ESP32-C3 that are reflected by the environment. The two chips are connected through a clock buffer to eliminate the relative frequency offset caused by the clock asynchrony between the two chips. In this way, the detection algorithm running on the ESP32-S3 can further utilize the phase information in the CSI data for detection. The calculation results and thresholds of the motion detection algorithm will be displayed on the screen in real time, and the motion detection results will be highlighted in red.



Our Patents

- A method and smart device for Wi-Fi human detection
- Method and STA device and AP device for performing multi-channel CSI scanning
- Wi-Fi Positioning Methods and Systems

2.3.6 ESP-IoT-Bridge Solution

ESP-IoT-Bridge Solution Introduction

ESP-IoT-Bridge Solution Overview This document will introduce the configuration process and usage of the ESP-IoT-Bridge solution. The ESP-IoT-Bridge solution is mainly aimed at the connection and communication between various network interfaces in IoT application scenarios, such as SPI, SDIO, USB, Wi-Fi, Ethernet, and other network interfaces. In this solution, the Bridge device can not only provide internet access for other devices but also serve as an independent device connecting to remote servers.

Solution Features:

The ESP-IoT-Bridge solution provides multiple network interfaces, which can be divided into two major categories:

- Interfaces for connecting to the internet
- Interfaces for helping other devices forward network data to connect to the internet

Users can implement personalized network interface connection solutions through various combinations of different network interfaces, maximizing the network advantages of Espressif chips.

The ESP chips that support the ESP-IoT-Bridge solution are as follows:

- ESP32
- ESP32-C3
- ESP32-S2
- ESP32-S3
- ESP32-C2
- ESP32-C6
- ESP32-C5

Common Application Scenarios of ESP-IoT-Bridge

ESP-IoT-Bridge: Wireless/Wired Router Solution:

• esp-iot-bridge/examples/wifi_router : The ESP-IoT-Bridge device connects to the router through Wi-Fi or a wired Ethernet port, and smart devices can access the internet by connecting to the SoftAP hotspot of the ESP-IoT-Bridge device.

Wireless NIC Solution:

• esp-iot-bridge/examples/wireless_nic : The ESP-IoT-Bridge device can be connected to a PC or MCU through multiple network interfaces (USB/ETH/SPI/SDIO). After a successful connection, a new network card will be added to the PC or MCU. Once provisioned successfully, it can provide internet access for the device.

Wired NIC Solution:

• esp-iot-bridge/examples/wired_nic : The ESP-IoT-Bridge device can connect to the network by plugging the Ethernet cable into the router's LAN port and connect to a PC or MCU through multiple network interfaces (USB/SPI/SDIO), providing internet access for the device.

SIM Card Solution:

- esp-iot-bridge/examples/4g_hotspot : The ESP-IoT-Bridge device can convert cellular networks into Wi-Fi signals after being equipped with a mobile network module with a SIM card. Nearby smart devices can connect to the internet after connecting to its hotspot.
- esp-iot-bridge/examples/4g_nic : The ESP-IoT-Bridge device can be equipped with a mobile network module with a SIM card. After the network module is connected to the internet, it can connect to a PC or MCU through multiple network interfaces (ETH/SPI/SDIO), providing internet access for the device.

ESP-IoT-Bridge Reference Materials In addition, we currently have some public ESP-IoT-Bridge solution software development materials and hardware materials, as follows:

- ESP-IoT-Bridge Software Reference
 - ESP-IoT-Bridge Solution Description
 - ESP-IoT-Bridge Solution Examples
- Development Board Purchase
 - ESP32-S3 Chip/Module Purchase

- ESP32-S3-USB-OTG Purchase
- Module/Development Board Materials and Selection Reference
 - Espressif Product Selection Tool

2.3.7 ESP-NOW Solution

Introduction to ESP-NOW Scheme

Overview of ESP-NOW Scheme The ESP-NOW protocol aims to provide direct communication between low-power, high-efficiency devices. It is suitable for IoT devices with ESP chips, which need to exchange data simply and quickly without connecting to traditional wireless networks (such as Wi-Fi routers).

The underlying dependency of the ESP-NOW application component is the connectionless communication protocol ESP-NOW defined by Espressif, which can directly, quickly, and low-power control smart devices without a router. It coexists with Wi-Fi and Bluetooth LE, supports multiple series of SoCs such as Espressif ESP8266, ESP32, ESP32-S, and ESP32-C. ESP-NOW is widely used in smart home appliances, remote control, and sensors. The features of ESP-NOW are as follows:

- Quick response
- Ultra-low power consumption
- High compatibility
- Long-distance communication
- Multi-hop control

Common Application Scenarios of ESP-NOW

- Espressif ESP-NOW wireless communication scheme: ESP-NOW is a wireless communication protocol based on the data link layer defined by Espressif, which can coexist with Wi-Fi and Bluetooth LE. It can not only achieve stable device connection and control but also serve as an independent auxiliary module to provide device provisioning, debugging, and firmware upgrade functions for the system.
- ESP32-C2 button battery switch: The low-power Wi-Fi button battery wireless switch solution based on ESP32-C2 can solve problems such as wireless switch response lag and the need for a gateway in Bluetooth LE, Zigbee, etc. It can be used with ESP-NOW without a gateway, with a sensitive response speed. The ESP32-C2 button battery switch not only supports ultra-long standby but also is not limited by the battery volume in product form.
- ESP32 Wi-Fi remote control: A Wi-Fi handheld remote control built based on ESP-NOW, supports instant control of smart devices equipped with ESP32, such as toy cars, small planes, robots, etc., can achieve millisecond-level control delay, and achieve a control distance farther than Bluetooth transmission.
- Batch provisioning
- Batch OTA
- Product production testing

How to Enable ESP-NOW ESP-NOW has a simple and clear enable process. Refer to the ESP-NOW get-started example to understand the enable process of ESP-NOW:

- 1. Call the espnow_storage_init() function to initialize the storage system of ESP-NOW. The ESP-NOW storage system is used to store the information of devices that have established communication, including the MAC address of the device, security keys, etc. This function initializes before starting to use ESP-NOW to ensure that the storage system is available.
- 2. Initialize Wi-Fi.

- 3. Create an espnow_config_t structure, which will be used to initialize ESP-NOW. And use the ESPNOW_INIT_CONFIG_DEFAULT() macro to initialize the structure. ESP-NOW_INIT_CONFIG_DEFAULT() is a macro provided by ESP-NOW to set default configuration parameters. Users can redefine this macro by undefining it.
- 4. Call espnow_init to initialize ESP-NOW.
- 5. Call the espnow_set_config_for_data_type() function to set the received data type, such as configuring it as ESPNOW_DATA_TYPE_DATA type. The second parameter is set to true, indicating that the data packet of this type is allowed to be received, and the third parameter is the pointer to the callback function. This means that when a data packet of type ESPNOW_DATA_TYPE_DATA is received, the callback function will be called to notify the received data.

ESP-NOW Reference Materials In addition, we currently have some public ESP-NOW materials, as follows:

- ESP-NOW github repository
- ESP-NOW User Guide
- ESP-NOW API Guide

ESP-NOW Performance Test Data Appendix This part can refer to the ESP-NOW User Guide.

2.3.8 Mesh Solutions

Comparison of Different Mesh Solutions

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Common Espressif Mesh solutions include:

- BLE Mesh : esp-ble-mesh doc
- Thread : esp-openthread doc
- Wi-Fi Mesh : esp-mesh-lite sdk
- Zigbee : esp-zigbee sdk

Note: In addition, ESP-Now can sometimes serve as a supplementary communication solution for the above Mesh solutions.

The following is a comparison table of key points of different Espressif Mesh solutions:

Feature	ESP BLE	ESP Thread	ESP-Mesh-	ZigBee Mesh	ESP-Now
	Mesh	Mesh	Lite	0	
Node count	Theoretically	Internal testing	Internal testing	Theoretical	Less than 100
	no more than	can reach 300	can reach 100	limit of 65000,	nodes
	32767, internal	nodes	nodes	internal testing	
	testing can			can reach 300	
	reach 100 nodes			nodes	
Communica-	Depends on the	The distance	Less than 100	The distance	Recommended
tion distance	packet sending	to maintain the	m, tested to	to maintain the	not more than
between nodes	power, it is	connection can	reach 170 m	connection can	150 m
	recommended	reach 200 m		reach 300 m	
	that the dis-				
	tance between				
	adjacent nodes				
	1s not more than				
These sharest	Jum	I arre a street to st	II'sh asteral test	I and a street to st	Lang lass than
Inrougnput	Low, less than 1	Low, actual test	High, actual test	Low, actual test	Low, less than
	Kops	1s about $4 - 3$	up to 20 Mops,	Is about 10.9	0.5 Mbps
		ND/S	the hierarchy	Kops	
			the lower the		
			throughput		
Single-hop	Within 100 ms	Average 10 ms	Within 100 ms	Average 16 ms	Within 10 ms
delav	at the limit dis-				
	tance of 50 m.				
	within 50 ms at				
	normal distance				
	(below 50 m)				

Table 24:	Mesh	Solution	Comp	arison
			1	

The core parameters of various Espressif Mesh solutions are explained below.

ESP BLE Mesh Advantages:

- Compatible with most devices, supports direct control and provision protocol standards from mobile phones
- Can access third-party devices that support BLE Mesh
- Supports low-power devices

Disadvantages:

• Data throughput is low, less than 1 Kbps

Core parameters are as follows:

- The chip is required to support BLE Mesh
- Supports coexistence of Wi-Fi & BLE Mesh & BLE ADV, but the device acting as a gateway should give as much RF time slice to BLE Mesh as possible, and the device acting as a node can relax this requirement appropriately
- Expected node count: theoretically no more than 32767, actually depends on the capability of the provisioner (gateway), using ESP32-S3 as the gateway, internal testing can reach 100 nodes
- Expected communication distance between nodes: depends on the packet sending power, it is recommended that the communication distance between adjacent nodes is not more than 50 m
- Coverage range: theoretically unlimited, actually depends on the uniform distribution of nodes
- Expected throughput per node: In the smart home field, scenarios with low data volume such as control signals, the heartbeat packet frequency of each node needs to be limited when there are many nodes
- Recommended chips: Nodes can use chips such as ESP32, ESP32-C3, ESP32-H2, and it is recommended to use the ESP32-S3 with PSRAM version for the gateway
- Single-hop delay: The delay at the extreme distance of 50 m is within 100 ms, and the delay at the normal distance (below 50 m) is within 50 ms
ESP Thread Mesh Advantages:

- Compatible with standard Thread devices, rich device networking functions
- IP-based communication mode (6LoWPAN and IPv6)
- Supports networking of hundreds of nodes (tested with 300 nodes)
- Fast networking speed, lower latency, self-recovery
- Supports low-power devices

Disadvantages:

Low data throughput, 802.15.4 MAC rate is 250 Kbps, UDP layer rate is 8 KB/s, actual test is about 4 - 5 KB/s

The core parameters are as follows:

- The chip is required to support 802.15.4 MAC, currently supported are ESP32-H2 and ESP32-C6
- Networking requires a Thread Border Router (OTBR) and several Thread nodes, any OTBR that complies with the Thread standard can be networked
- Expected number of nodes: The theoretical value can be hundreds or more, internal testing shows that 300 nodes can be networked successfully
- Expected communication distance between nodes: Depends on the packet sending power, under the condition of H2 Rx sensitivity of -102.5 dBm, txpower 20 dBm, the distance to maintain the connection can reach about 200 m
- Coverage: Theoretically unlimited, actually depends on the uniform distribution of nodes
- Expected throughput per node: In the smart home field, scenarios with low data volume, suitable for simple control, sensor temperature collection and other IoT fields
- Recommended chips: Nodes can use chips such as ESP32-H2 and ESP32-C6, OTBR is recommended to use ESP32-S3 with PSRAM version + ESP32-H2 for receiving and sending 802.15.4 messages
- Sub-node communication: Each sub-node can communicate with each other, and if the OTBR can go online and enables NAT64, the sub-node can go online through the OTBR. It has advantages for scenarios where sub-nodes need to interact with the cloud and the throughput demand is small
- Single-hop delay: Average 10 ms

ESP-Mesh-Lite Advantages:

- High throughput, the shielded box test results show that the maximum throughput of the second layer node interaction can reach 20 Mbps, decreasing layer by layer, with the highest at the fifth layer being 6 Mbps.
- Connection stable, supports other non-ESP devices to communicate with devices in Mesh-lite via Wi-Fi.
- No need for a gateway, can directly access the Internet

Other points:

• ESP-Now function has been integrated

The core parameters are as follows:

- Expected number of nodes in a single Mesh network: less than 100
- Expected communication distance between nodes: less than 100 m, if the data throughput requirement is not high, it can reach 170 m through testing
- Supported levels: Currently supports up to 15 levels, recommended 5 to 6 levels
- Expected range of a single Mesh network: Theoretically unlimited, the actual range is mainly limited by the maximum number of nodes, the maximum level, and the maximum communication distance between nodes
- Single-hop delay: Within 100 ms under normal test environment

ZigBee Mesh Advantages:

- Compatible with standard ZigBee nodes
- · Fast networking speed, low latency, self-recovery
- Supports low-power devices

Disadvantages:

• Low data throughput, 802.15.4 MAC rate is 250 Kbps, ZigBee rate is 16.9 Kbps

The core parameters are as follows:

- The chip is required to support 802.15.4 MAC, currently ESP32-H2 and ESP32-C6 can be chosen
- Networking requires a Coordinator and several ZigBee nodes (routers or end nodes), any that comply with the ZigBee standard can be networked
- Expected number of nodes: The theoretical upper limit is 65000, ultimately depends on how long each node needs to communicate on the network, and how much data loss or retransmission the application can tolerate. Internal testing shows that 300 nodes can be stably networked, point-to-point communication
- Expected communication distance between nodes: Depends on the packet sending power, under the condition of H2 Rx sensitivity of -102.5 dBm, txpower 20 dBm, the distance to maintain the connection can reach about 300 m
- Coverage: Theoretically unlimited, actually depends on the uniform distribution of nodes
- Expected throughput per node: In the field of smart homes, control signals and other low data volume scenarios, it is suitable for simple control, sensor temperature collection and other IOT fields
- Recommended chips: Nodes can use chips such as ESP32-H2 and ESP32-C6, and it is recommended to use ESP32-S3 with PSRAM version + ESP32-H2 for receiving and sending 802.15.4 messages as Coordinator
- Single hop delay: Average 16 ms

ESP-Now Advantages:

• Quick response, low power consumption

Disadvantages:

• Small throughput

The core parameters are as follows:

- Current ESP chip/module selection: All ESPs that support Wi-Fi are available
- Whether BLE function is needed at the same time: It can coexist with BLE, and there will be power consumption requirements when it involves BLE coexistence
- Expected communication method: ESP-NOW has two communication methods, unicast and broadcast. Mesh often recommends the form of broadcast, and the subsequent performance data, if not specified, are all based on the form of broadcast
- Estimated number of nodes: Less than 100
- Estimated communication distance between nodes: Less than 150 m is relatively stable, and the limit distance in open environment is 300 m (point-to-point test, broadcast distance is further)
- Expected throughput per node: Must be less than 0.5 Mbps
- Expected communication delay: Less than 10 ms within a communication distance of 150 m
- Whether there is a requirement for low power consumption: If there is a requirement for low power consumption, the estimation method is consistent with the Wi-Fi auto light sleep (power save) scenario, and you can also refer to the coin cell demo
- Whether it needs to connect to a Wi-Fi router: It involves channel switching, and it needs to confirm whether it supports channel switching after connecting to the router according to the IDF version
- Whether it needs to coexist with upper layer communication protocols (such as TCP/UDP, etc.): The recommended use scenario of ESP-NOW is LAN. If it involves upper layer transmission protocols (TCP, etc.), it is recommended to use Thread

Introduction to ESP-Mesh-Lite Solution

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Overview of ESP-Mesh-Lite Solution ESP-Mesh-Lite is a Mesh solution built on the Wi-Fi protocol, based on the SoftAP + Station mode, and modeled after IoT-Bridge. ESP-Mesh-Lite allows a large number of devices distributed

over a wide area (both indoors and outdoors) to connect with each other within the same WLAN (Wireless Local Area Network). The advantages of this solution are as follows:

- ESP-Mesh-Lite is a network structure different from traditional Wi-Fi networks. In the ESP-Mesh-Lite network, nodes do not need to connect to a central node, but can directly connect with neighboring nodes. This interconnection between nodes allows for a wider network coverage area, unrestricted by the distance to the central node.
- Traditional Wi-Fi networks usually adopt a centralized structure, where all nodes connect to a central node (such as a router). This network structure may cause the central node to be capacity-limited, affecting the scalability and stability of the network. However, in the ESP-Mesh-Lite network, nodes are interconnected, allowing nodes in the network to be responsible for data forwarding of neighboring nodes, no longer limited by the capacity of the central node.
- Since each node in the ESP-Mesh-Lite network is assigned an IP address by the parent node, nodes can access the network as if they were individual devices connected to a router. The transmission of information is imperceptible to the parent node, greatly reducing the difficulty of application layer development.

In summary, the ESP-Mesh-Lite network has a tree-like network structure, strong scalability, wide coverage, interconnected nodes, and increased network flexibility. Compared with traditional Wi-Fi networks, ESP-Mesh-Lite has greater advantages and applicability in IoT applications.

Common Application Scenarios of ESP-Mesh-Lite

- Smart home control systems: Due to the interconnection between nodes, smart devices can communicate directly with neighboring devices, improving response speed and stability.
- Smart agriculture: Many wireless sensors in the fields need to communicate with each other and send messages to the cloud.
- Photovoltaic inverters: Upload collector data to the cloud.
- Smart lighting: Group control and group control of lights and data reporting.
- Environmental monitoring: Can achieve data collection and transmission of multiple environmental monitoring devices, providing more comprehensive environmental data.
- Education and training: Connect multiple teaching devices and student terminals to achieve sharing and interaction of teaching resources.
- Smart security: Multiple security devices can achieve data sharing and linkage control, improving the response speed and reliability of the security system.
- Remote device management: Perform OTA firmware upgrades for all devices in the network.

Reference Materials for Getting Started with ESP-Mesh-Lite ESP-Mesh-Lite SDK:

• ESP-Mesh-Lite SDK

ESP-Mesh-Lite Solution Guide Document: (Must-read, provides an introduction to the ESP-Mesh-Lite protocol)

- Chinese Document
- English Document

ESP-Mesh-Lite Basic Example Link: (It is recommended for developers to run this example to simply understand the basic usage of ESP-Mesh-Lite and related APIs)

• Basic Example Link

ESP-Mesh-Lite + Rainmaker Example Link: (Includes provision, LAN OTA, Mesh node communication and other advanced usage)

• Rainmaker Example Link

ESP-Mesh-Lite Component Manager Link:

Component Manager

How to Enable ESP-Mesh-Lite Enabling ESP-Mesh-Lite is very convenient, developers only need to simply call some API commands to enable ESP-Mesh-Lite. Refer to the mesh_local_control example:

- 1. Initialize the NVS partition of ESP32 to save configuration information.
- 2. Initialize the network interface. Call the *esp_netif_init()* function.
- 3. Create the default event loop. Call the *esp_event_loop_create_default()* function.
- 4. Create all network interfaces, including STA mode and SoftAP mode. Call the *esp_bridge_create_all_netif()* function.
- 5. Initialize Wi-Fi functions, including initializing Wi-Fi' s STA mode and SoftAP mode, first connect to the specified Wi-Fi hotspot (STA mode), and then open the SoftAP hotspot (SoftAP mode). When connecting to Wi-Fi and opening SoftAP, try to read the previously saved configuration from NVS. If the reading fails, use the predefined default value.

Station:

- Define a *wifi_config_t* structure variable *wifi_config* to configure the parameters of Wi-Fi' s STA mode.
- Set the ssid (Wi-Fi hotspot name) and password (Wi-Fi hotspot password) of wifi_config.sta.
- Set the Wi-Fi mode to STA mode, and use *wifi_config.sta.ssid* and *wifi_config.sta.password* as parameters to connect to the specified Wi-Fi hotspot. The fourth parameter is NULL, indicating that no callback function is set. Call the *esp_bridge_wifi_set* function.

SoftAP:

- Clear the content of the wifi_config structure to prepare for the configuration of SoftAP mode below.
- Define a variable to store the SSID length in SoftAP mode.
- Get the SSID in SoftAP mode from NVS. If the acquisition fails, use the default value defined by the *CONFIG_BRIDGE_SOFTAP_SSID* macro. Call the *esp_mesh_lite_get_softap_ssid_from_nvs* function.
- Define a variable to store the password length in SoftAP mode.
- Get the password in SoftAP mode from NVS. If the acquisition fails, use the default value defined by the *CONFIG_BRIDGE_SOFTAP_PASSWORD* macro. Call the *esp_mesh_lite_get_softap_psw_from_nvs* function.
- Set the Wi-Fi mode to SoftAP mode, and use *wifi_config.ap.ssid* and *wifi_config.ap.password* as parameters to open the SoftAP hotspot. The fourth parameter is NULL, indicating that no callback function is set. Call the *esp_bridge_wifi_set* function.
- 6. Provide default values for Mesh network configuration. Define the *esp_mesh_lite_config_t* structure variable and initialize it with the *ESP_MESH_LITE_DEFAULT_INIT()* macro.
- 7. Initialize ESP-Mesh-Lite and start the Mesh-Lite network. Call the *esp_mesh_lite_init()* function and configure it with the structure defined in the previous step.

ESP-Mesh-Lite Performance Reference Appendix

- Common performance indicators of ESP-Mesh-Lite
- Comparison of different Mesh solutions

2.3.9 Provisioning Solution

Provisioning Scheme Introduction

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Attention: The provisioning scheme in this document refers to the scheme for configuring and connecting ESP devices to Wi-Fi in different ways. Other schemes such as BLE-MESH provisioning are not included in this document.

Scheme Overview During the code development phase, Wi-Fi information (SSID and PASSWORD) is often written directly into the program, which cannot be changed at any time to connect to the desired Wi-Fi, which is obviously impractical. Therefore, we have developed and launched a series of provisioning schemes:

- SoftAP Provisioning
- Blufi Provisioning
- Smartconfig Provisioning
- WEB Provisioning

Taking the common application scenario of provisioning smart sockets as an example, as follows.



Allow users to configure Wi-Fi information to ESP devices through mobile phones. Different provisions have their own advantages and disadvantages, and the next section elaborates on the details of these provisioning schemes.

Common Provisioning Application Scenarios

1. Blufi Provisioning (Requires ESP chip to support BLE) The ESP device will broadcast BLE, and the nearby mobile phone will ask the user whether to connect to the ESP device via BLE after receiving the broadcast. If the connection is selected, the mobile phone can send the Wi-Fi information that the ESP device needs to connect to the ESP device. In this process, the user does not need to switch Wi-Fi networks, but needs to turn on the mobile phone's Bluetooth. The user experience is relatively better than SoftAP provisioning. However, it is necessary to add Bluetooth-related code on the device side, which will increase the size of the firmware and occupy a certain memory before the provisioning is completed.

Software Code & APP Reference Links:

- Blufi Provisioning Example
- Android APP & Source Code
- iOS APP
- iOS APP Source Code
- BluFi Protocol

- BluFi Provisioning Guide
- Blufi WeChat Provisioning Mini Program Corresponding Source Code

2. Smartconfig Provisioning This method does not require the establishment of any communication link. The mobile phone sends UDP broadcast packets of different lengths to represent Wi-Fi information. The ESP device listens to all data frames within the signal coverage range in promiscuous mode and obtains Wi-Fi information through a certain algorithm. The disadvantage is that the success rate of provisioning is greatly affected by the environment.

Software Code & APP Reference Links:

- Smartconfig Provision Example
- Android APP & Source Code
- iOS APP
- iOS Source Code

3. SoftAP Provision The ESP device will establish a Wi-Fi hotspot, and after the user connects their phone to this hotspot, they send the Wi-Fi information that the ESP device needs to connect to. This provisioning mode requires the user to manually connect to the hotspot network of the ESP device, which makes the provisioning process complicated, but this method generally has a higher success rate than Smartconfig provisioning.

Software Code & APP Reference Links:

- SoftAP Provision Example
- Android APP
- iOS APP
- Android APP Source Code
- iOS APP Source Code

4. WEB Provision (Currently not recommended, and there are no mature examples) Establish a hotspot on the ESP device, connect with the phone, and open the configuration page in the browser to complete the provision. This method is very reliable and allows provisioning to be completed on the computer. The disadvantage is that it requires space on the device to embed the webpage.

This solution is currently not recommended. If you still need code references, you can refer to WEB Provision Example. Please note that this example is very old and it is not guaranteed whether it can be executed normally on the new version of ESP-IDF. It only has some reference value in terms of code logic.

The table below summarizes the key parameters of the four provisioning schemes mentioned above.

Provisioning	Need Extra	Need Extra	Provisioning	Operation	Recommen-
Scheme	BLE	Mobile APP	Success Rate	Complexity	dation Level
Blufi Provision	✓	\checkmark	High	Simple	Recommended
Smartconfig	×	\checkmark	Relatively High	Simple	Medium
Provision					
SoftAP Provi-	×	✓	High	Complex	Medium
sion					
WEB Provision	×	×	High	Complex	Medium
(Currently not					
recommended)					

Table 25: Provisioning Scheme Summary Table

Reference Materials The brief summary of the provisioning scheme materials is as follows:

- Blufi Provision Example
- Smartconfig Provision Example
- SoftAP Provision Example

Attention: If you need more references like provision APP, please get the corresponding information in *Common Provisioning Application Scenarios*.

2.3.10 Thread Solution

Thread Solution Introduction

Thread Solution Overview The Thread solution is mainly aimed at low-power, low-bandwidth devices in IoT application scenarios. It is a low-power, wireless, mesh technology solution based on IPv6 running on 802.15.4 radio technology.

You can also view the following video demos for a more intuitive understanding of this type of solution:

- [Espressif Demo] | Demonstration of Large-Scale Thread Network Provisioning Performance Based on Espressif Thread Chip
- Espressif Thread Border Router Solution

Solution Advantages:

- Based on IPv6, it supports flexible settings, monitoring, data analysis, and configuration, and can directly connect the local network to the cloud without sacrificing encryption and security.
- In terms of application protocol selection, it provides flexibility, even allowing multiple application standards to run simultaneously, making it more flexible to integrate with other technologies.
- Compatible with standard Thread devices, rich device provisioning features.
- Supports hundreds of node network topologies.
- Fast provisioning speed, low latency, self-recovery.
- Supports low-power devices

Common Thread Application Scenarios

- Smart Home Control Systems: Due to the interconnection between nodes, smart devices can communicate directly with neighboring devices, improving response speed and stability.
- Smart Agriculture: Many wireless sensors in the fields need to communicate with each other and send messages to the cloud.
- Photovoltaic Inverters: Upload collector data to the cloud.
- Smart Lighting: Group control and group control of lights and data reporting.
- Environmental Monitoring: Can realize data collection and transmission of multiple environmental monitoring devices, providing more comprehensive environmental data.
- Remote Device Management: Perform OTA firmware upgrades for all devices in the network.

Thread Reference Materials In addition, we currently have some public Thread materials, as follows:

- esp-thread-br SDK github repository
- Thread Introduction
- Thread API

2.3.11 Zigbee Solution

ZigBee Solution Introduction

ZigBee Solution Overview The ZigBee solution mainly targets the connection and communication between various network interfaces in IoT application scenarios, such as SPI, SDIO, USB, Wi-Fi, Ethernet, and other network interfaces.

Features of the solution:

- Compatible with standard ZigBee nodes
- Fast network formation, low latency, self-recovery
- Supports low-power devices

The ESP chips that support the ZigBee solution are as follows:

- ESP32H2
- ESP32C6

Common ZigBee Application Scenarios

ZigBee GateWay: NCP Solution:

• esp-zigbee-sdk/examples/esp_zigbee_ncp : Demonstrates how to configure a ZigBee network co-processor device, accessed via UART to the MCU. The MCU device connects to the router via Wi-Fi or wired Ethernet, and smart devices achieve network control by connecting to the device coordinator.

RCP Solution:

• esp-idf/examples/openthread/ot_rcp : Demonstrates how to configure an 802.15.4 radio, accessed via UART to the Espressif SoC. The MCU device connects to the router via Wi-Fi or wired Ethernet, and smart devices achieve network control by connecting to the device coordinator.

ZigBee Standard Node: Coordinator Solution:

• esp-zigbee-sdk/examples/esp_zigbee_HA_sample/HA_on_off_switch: Demonstrates how to configure a Zig-Bee coordinator and use it as an HA On Off switch device.

End Device Solution:

• esp-zigbee-sdk/examples/esp_zigbee_HA_sample/HA_on_off_light: Demonstrates how to configure a ZigBee end node and use it as an HA light device.

How to Enable esp_zigbee_sdk esp_zigbee_sdk has a simple and clear enablement process. Refer to the esp_zigbee_sdk esp_zigbee_HA_sample example to understand the enablement process of esp_zigbee_sdk:

- 1. Call the nvs_flash_init() function to initialize the storage system of esp_zigbee_sdk.
- 2. Call the esp_zb_platform_config() function to initialize the device communication method.
- 3. Call the esp_zb_init() function to initialize the ZigBee node role.
- 4. Call the esp_zb_on_off_switch_ep_create() function to create the Cluster and Attribute required by the application layer.
- 5. Call the esp_zb_device_register() function to register the Cluster and Attribute processed by the application layer.
- 6. Call the esp_zb_start() function to start the ZigBee protocol stack.
- 7. Call the esp_zb_main_loop_iteration() function to handle the events reported by the ZigBee protocol stack.

ZigBee Reference Materials In addition, we currently have some public ZigBee solution software development materials and hardware materials, as follows:

- ZigBee Software Reference
 - ZigBee Solution Description
 - ZigBee Solution Examples
- Development Board Purchase
 - ESP32-H2 & ESP32C6 Chip/Module Purchase
 - ESP32-H2-DevKitM-1 Purchase
 - ESP32-C6-DevKitM-1 Purchase
- Module/Development Board Materials and Selection Reference
 - Espressif Product Selection Tool

2.3.12 Wireless Network Card Solution

Comparison of Different Wireless Network Card Solutions

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Most Espressif SoCs can serve as Wireless Network Interface Controllers (WNICs), enabling other IoT devices and MCUs to access the internet. The ESP-WNIC solution provides just such functionality. Users can choose the Espressif SoC that best suits their application scenario, thereby benefiting from the low development cost, easy maintenance, and highly scalable features offered by Espressif SoCs. Through the ESP-WNIC solution, users can quickly join a wireless network, connect to the cloud platform, transmit data, and achieve remote control.

Common Espressif wireless network card solution SDKs include:

- esp-at
- esp-hosted
- esp-iot-bridge
- usb dongle

Below is a comparison table of different Espressif wireless network card solutions:

Network Card So-	ESP-AT	ESP-Hosted	ESP-IoT-Bridge	USB-Dongle
lution				
MCU Adaptation	Easy	Medium	Medium	Easy ¹
Difficulty				
MCU Workload in	Light	Medium	Medium	Medium
Complete Solution				
MCU Resource Re-	Ability to parse	Ability to run	Ability to run	Ability to support
quirements	strings	TCP/IP network	TCP/IP network	USB host and
		stack	stack	TCP/IP network
D 1 1	L' DTOG N	L' DEOG	L' DTOG	protocol stack
Recommended	Linux, RTOS, Non-	Linux, RTOS	Linux, RTOS	Linux, MacOS,
MCU System	US			Windows
Hardware Commu-	UART, SDIO, SPI,	SDIO, SPI	USB, ETH, SPI,	USB
nication Interface	USB		SDIO	
Recommended	$0 \sim 10$ Mbps	$0 \sim 20$ + Mbps	$0 \sim 25$ Mbps	$0 \sim 6$ Mbps
Throughput Range				
Typical Application	Smart home, con-	Robots, set-top	Smart home, con-	Robots, set-top
Scenarios	sumer electronics,	boxes, remote	sumer electronics,	boxes, etc.
	POS machines,	video, etc.	IPC, low power	
	advertising screens,		scenarios requiring	
	remote data col-		MCU sleep, etc.	
	lection, LBS			
	positioning, low			
	power scenarios			
	requiring MCU			
	sleep, etc.			
Resource Links	esp-at	esp-hosted	esp-iot-bridge ²	usb dongle

Table 26:	WNIC	Solution	Comparison
-----------	------	----------	------------

Note:

- ¹ : The MCU adaptation difficulty of the USB-Dongle is easy, but please note that the MCU needs to support systems with standard USB ECM/NCM protocol stacks such as Linux, MacOS, or Windows.
- ² : If access to esp-iot-bridge Github is difficult, you can alternatively visit esp-iot-bridge component for further understanding.

The complete explanation of the advantages and disadvantages of various Espressif wireless gateway solutions is as follows.

ESP-AT Advantages:

- No development required on the ESP side
- Low MCU resource requirements
- Suitable for various MCU systems, such as Linux, RTOS, non-operating systems

Disadvantages:

• Lower data throughput (0~10 Mbps)

Reference materials:

- Official SDK: esp-at
- Official documentation: ESP-AT User Guide
- Chinese blog: Espressif ESP-WNIC Wireless Network Card Solution
- Chinese video: Espressif ESP-WNIC Wireless Network Card Solution

ESP-Hosted Advantages:

- Suitable for various MCU systems, such as Linux, RTOS
- High data throughput, up to 25 Mbps with ESP32-C6 and SDIO interface, for specific throughput test results, refer to Throughput performance

Disadvantages:

- The MCU needs to have the ability to run the TCP/IP network stack
- High MCU adaptation difficulty and workload
- ESP as a peripheral of the MCU, cannot work independently and completely

Reference materials:

- Official SDK: esp-hosted
- Official documentation: ESP-Hosted Readme
- Chinese blog: Espressif ESP-WNIC Wireless Network Card Solution
- Chinese video: Espressif ESP-WNIC Wireless Network Card Solution

ESP-IOT-BRIDGE Advantages:

- Supports multiple peripherals such as ETH, SPI, SDIO, USB, and can be extended to support ESP Mesh Lite Solution
- High data throughput, using ESP32 and SDIO interface can reach 25 Mbps
- Low development cost for ESP and main MCU
- Can independently run some application functions, providing conditions for low power scenarios such as MCU sleep

Disadvantages:

• MCU needs to have the ability to run TCP/IP network stack

References:

- Official SDK: esp-iot-bridge
- Official documentation: ESP-IoT-Bridge Solution
- Chinese blog: Espressif ESP-IoT-Bridge Solution Supports Flexible Device Provisioning
- Chinese video: Espressif ESP-IoT-Bridge Networking Solution

USB-Dongle Advantages:

- Low adaptation difficulty on MCUs supporting USB host ECM/NCM Class (such as Linux, Windows, MacOS)
- High data transmission stability (using USB interface)

Disadvantages:

- Relatively low data throughput (0~6 Mbps)
- Limited application scenarios (such as robots, set-top boxes, etc.)
- ESP as a peripheral of MCU, cannot work independently and completely

References:

- Official example: usb dongle
- Official documentation: USB-Dongle Readme

2.3.13 Lighting Solution

Introduction to Lighting Solutions

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

Overview and Advantages of Light Solutions Espressif's lighting solutions are renowned for their richness and maturity. Our solutions support a variety of common lamp-related peripherals, including LEDC, SPI, I2C, MCPWM, etc., offering a wide range of choices for your lighting needs. Combined with Espressif's mature wireless protocols such as Wi-Fi and BLE, as well as cloud adaptation technology, you can easily implement remote control functions and manage your lighting system anytime, anywhere.

Moreover, Espressif has implemented AI voice recognition technology, allowing you to control lights through voice. Our audio algorithm makes music rhythm lights possible, while color recognition and image rhythm algorithms provide you with a variety of cutting-edge innovative solutions. These innovative technologies will bring more possibilities to your lighting system, making it smarter, richer, and more attractive. Here are the advantages of Espressif's lighting solutions:

- Zero-code lighting solution: ESP ZeroCode Solution No need to worry about complex steps such as firmware, mobile APP, cloud connection, certification, and production, directly build smart home devices compatible with Matter, including lighting devices
- Rich software references: Provide comprehensive lamp-related software development materials, including detailed guidance documents and examples

Common Application Scenarios of Light Espressif's lighting solutions are widely used in various fields, and the types of lamps used in different fields also vary, summarized as follows:

	ESP32	ESP32-S3	ESP32-C2	ESP32-C3	ESP32-C6
General					
Lighting					
Bulb/Desk	LEDC/I2C	LEDC/I2C	LEDC/I2C	LEDC/I2C	LEDC/I2C
Lamp					
Ad-	SPI	SPI/RMT	SPI	SPI/RMT	SPI/RMT
justable					
Color					
Tem-					
perature					
Strip	LEDC	LEDC	LEDC	LEDC	LEDC
Spot-	LEDC	LEDC	LEDC	LEDC	LEDC
lignt/Downi	ignt/Candlelignt/Cel	ing			
Ligin					
Ambient					
Lighting					
Colorful	SPI	SPI/RMT	SPI	SPI/RMT	SPI/RMT
Strip/String	~				
Lights					
Copper	SPI	SPI	SPI	SPI	SPI
Wire					
Lights					
Aroma	SPI	SPI/RMT	SPI	SPI	SPI/RMT
Lamp/Splic	ing				
Lamp					
Other					
Acces-					
sories					
Silicon	MCPWM	МСРWМ	X	X	MCPWM/EIM+Analog
trolled					Comparator
Dimmer					
Diminer					
Other					
Func-					
tions					
Music	\checkmark	\checkmark	\checkmark	\checkmark	\checkmark
Rhythm					
ESP-	\checkmark	\checkmark	\checkmark		\checkmark
NOW					
Local					
Control					
CSI	\checkmark	\checkmark	\checkmark	$ $ \checkmark	\checkmark
Human					
Detection					
Function					
Infrared	Х	\checkmark	Х	√	√
Remote					
Control					

Light Reference Materials In addition, we currently have some public light github repositories, videos, and module materials, as follows:

• Software Reference

- ESP-IDF LEDC Example
- ESP-IDF RMT LED STRIP Example
- Rhythm Flowing Light Example
- Module/Development Board Materials and Option Reference
 - Espressif Product Selection Tool

Light Dimming Solutions Summary

PWM Dimming Solution PWM dimming is a technique to control LED brightness by adjusting pulse width, the core of which is to adjust by changing the duty cycle of the current pulse (i.e., the proportion of high-level time in the entire cycle time). When the duty cycle is high, the LED gets a long current time and high brightness; conversely, when the duty cycle is low, the LED gets a short current time and low brightness. All ESP chips support using hardware LEDC Driver / MCPWM to output PWM, with the LEDC driver recommended for implementation first, as it supports hardware gradient, configurable frequency duty cycle, and a maximum resolution of 20 bit.

Note:

I2C Dimming Solution I2C Dimming sends control commands to the dimming chip through the I2C bus, adjusting the LED brightness by changing the output current of the dimming chip. The I2C bus consists of two lines: the data line (SDA) and the clock line (SCL). All ESP chips support using hardware I2C dimming chips, and currently, perfect driving has been completed on common dimming chips such as SM2135EH, SM2235EGH, SM2335EGH, BP5758, BP5758D, BP5768, BP1658CJ, KP18058.

Note:

- 1. How much should the pull-up resistor for the I2C driver be selected?
 - Usually choose between 2.2 k-4.7 k, the signal quality can be determined by actual measurement.

Single-line Dimming Solution The single-wire dimming solution is a method of controlling LED brightness through a single communication line, the core of which is to adjust the brightness of the LED by sending control signals through a specific communication protocol. There are currently 2 categories:

- WS2812 and SK6812 types use a separate data bus to transmit color data through specific timing control.
- Similar to power line carrier, multiplexing VCC and modulating communication data on it, only 2 wires are needed for the bead.

This type of solution can be implemented on the ESP chip using RMT peripherals or SPI peripherals. It is recommended to use SPI for bead communication control, and RMT is mostly used for receiving infrared remote control data.

Note:

- 1. What is the principle of SPI implementation?
 - Taking WS2812 as an example, it has RGB three-channel data, each channel is 1 byte, the default configuration SPI clock is 2.4 MHz, single bit transmission, then sending a bit is a clock about 400 ns, usually use 3 bits to represent 0 code and 1 code, that is, 0 code is composed of 2 bit high + 1 bit low, 1 code is composed of 1 bit high + 2 bit low, so each bit of 1 byte original data needs to be expanded to 3 bits to represent 0 and 1, so 1 byte original data needs to be expanded to 3 bytes SPI data. If we send orange, then the conversion is as follows:

Does it support outputting complementary waveforms to adjust color temperature and brightness?
 Yes, the hpoint function of LEDC can be used for phase shifting.

255 (0b1111111)	->	11011011	01101101	10110110	->	0xDBL
→UX6D UXB6						
127 (0b01111111)	->	10011011	01101101	10110110	->	0x9BL
→0x6D 0xB6						
- OKOD OKDO						
0 (0b0000000)	->	10010010	01001001	00100100	->	0x92_
⊶0x49 0x24						

The actual data sent by SPI is $\{0xDB \ 0x6D \ 0xB6\}, \{0x9B \ 0x6D \ 0xB6\}, \{0x92 \ 0x49 \ 0x24 \}$, the RGB data sequence of different light strips may be different, and need to be adjusted according to the actual situation.

- 2. Does SPI support connecting multiple light strips?
 - Yes, changing the SPI transmission mode to 4 bit can support 4 light strips. It should be noted that the original data conversion to SPI data also needs to be changed. At this time, 4 bits of data will be sent in 1 clock.
- 3. Can you provide the design idea of SPI driving light strip driver?
 - If the SPI clock is 2.4 MHz, each clock is 0.4167 us, if it is sent in 8 bit mode, 1 byte is sent in 1 clock, 10 KB is also 10000 clocks, and it is sent in 4160 us. Use the STC mode of SPI, and group 10 1 KB brush light data in one spi_multi_transaction_t. The SPI queue length is set to 1, malloc 2 10 KB buf, each time the data is grouped, use spi_device_queue_multi_trans to send, then another buf fills the data, after filling, use spi_device_get_multi_trans_result to get the previous result of the light strip, after returning successfully, group the data again, and 2 buf cycle fill the data.

Silicon Controlled Dimming Solution Triac dimming is a dimming technology that controls the brightness of a lamp by adjusting the phase of the current. It is widely used in incandescent lamps, halogen lamps, and some LED lamps compatible with triac dimming. The core is to adjust the phase angle of the triac conduction, changing the conduction time of the current in each AC cycle, to achieve the regulation of the lamp' s power output and brightness. There are mainly two types of phase-cutting methods for triac dimming:

- Leading edge cutting: Cutting is performed in the first half of the AC waveform, and the Triac starts to conduct at the leading edge of each AC cycle. Suitable for incandescent lamps and some compatible LED lamps.
- Trailing edge cutting: Cutting is performed in the second half of the AC waveform, and the Triac starts to conduct at the trailing edge of each AC cycle. Suitable for some compatible LED lamps and electronic transformers.

The software design follows the following 3 steps:

- 1. Phase detection: Detect the zero-crossing point of the AC power to determine the timing of the dimming signal. Zero-crossing detection can be achieved through optocouplers and detection circuits.
- 2. Phase control: Calculate the corresponding conduction delay time according to the brightness set by the user, and control the Triac to conduct at the appropriate time.
- 3. Trigger control: After detecting the zero-crossing point, trigger the Triac to conduct according to the set delay time, generally using a 50 us pulse to trigger conduction.

In ESP chips, the above functions are usually implemented using MCPWM peripherals or ETM + Analog comparator peripherals. It is recommended to use MCPWM peripherals for implementation. The implementation principle is: Enable 2 comparators A and B on the same timer, and operate the same IO. Comparators A and B output low levels when the timer counter is 0. Set a comparison value x, x is the delayed conduction time, and set comparator A to output a high level when the timer is x, and comparator B to output a low level when the timer is x + 50. Use the synchronization function of MCPWM to capture the zero-crossing point. The timer will be automatically reset when synchronized, thus completing a phase-cutting control. The accuracy of the zero-crossing point capture can be controlled to less than 1 us.

Note:

- 1. Can it be implemented on chips such as ESP32C2 that do not have MCPWM peripherals?
 - It can be implemented using a gptimer + IO simulation scheme. All control functions need to be placed in ram. However, due to chip and software design factors, the accuracy of zero-crossing point capture will still be affected by system interrupts. Especially after enabling Wi-Fi and Bluetooth, it usually fluctuates

between 4.7 us \sim 20 us, which is very unstable. This will cause poor compatibility of the dimmer, and some LED lamps compatible with the dimmer will have intermittent flickering. If the evaluation is acceptable, the above scheme can be used.

Light Additional Feature Components

Music Rhythm Music rhythm is a lighting control technology that analyzes music signals in real time to synchronize the lights with the rhythm, melody, and sound effects of the music. All ESP32 platform chips support local music rhythm, obtain audio data through ADC, and then get waterfall, jump, gradient and other lighting effect data after FFT or loudness calculation, refresh to the bulb through the driver, and display.

- Example code
- Demo video

ESP-NOW Local Control Function ESP-NOW is a connectionless Wi-Fi communication protocol defined by Espressif. In ESP-NOW, application data is encapsulated in the action frames of various vendors, and then transmitted from one Wi-Fi device to another Wi-Fi device without connection. In the field of smart lighting, this technology is often used as a random sticker switch for lamps.

- Example Code
- Demonstration Video

CSI Human Detection Function Channel State Information (CSI) is an important parameter describing the characteristics of wireless channels, including signal amplitude, phase, signal delay, and other indicators. In Wi-Fi communication, CSI is used to measure the state of wireless network channels. CSI is very sensitive to environmental changes. It can not only perceive large-scale movements such as walking and running of people or animals, but also perceive subtle movements such as breathing and chewing in a static environment. In the field of smart light bulbs, CSI can be used as a human perception sensor to realize the function of turning on the light when someone comes and turning off the light when someone leaves.

- Example Code
- Demonstration Video

Infrared Remote Control Infrared remote control is a technology that uses infrared light signals to transmit control instructions, which is widely used in the remote control operation of household appliances such as televisions, air conditioners, audio equipment, and lighting fixtures. The infrared remote control system usually consists of an infrared transmitter (remote control) and an infrared receiver (controlled device). The infrared transmitter modulates the infrared light signal and sends out the control instruction in the form of pulse coding. After the infrared receiver receives the signal, it demodulates and decodes the control instruction, and then performs the corresponding operation. In the lighting field, this technology is used as a local infrared remote control for light strips, downlights, spotlights, and ambient lights.

• Example Code

Light Bulb Driver Component The Light Bulb Component encapsulates several common dimming schemes in light bulbs, and adds effects, calibration, state memory, power limit and many other common functions, making it easy for developers to integrate into their own applications. All ESP32 series chips are currently supported.

Supported dimming schemes are as follows

- PWM scheme
 - RGB + C/W
 - RGB + CCT/Brightness
- I2C dimming chip scheme
- SM2135EH

- SM2X35EGH (SM2235EGH/SM2335EGH)
- BP57x8D (BP5758/BP5758D/BP5768)
- BP1658CJ
- KP18058
- Single bus
 - WS2812

Supported lamp board layouts are as follows

- Single channel: Cold or warm color temperature beads, can complete brightness control under single color temperature.
- Dual channel: Cold and warm beads, can complete color temperature and brightness control.
- Three channels: Red, green, and blue beads, can complete any color control.
- Four channels: Red, green, blue, cold or warm beads, can complete color and single color temperature brightness control, if a color mixing table is configured: it supports using these beads to mix different color temperatures, to achieve color temperature control.
- Five channels: Red, green, blue, cold, warm beads, can complete color and color temperature brightness control.

Gradient Principle The gradient in the light bulb component is implemented by software. Each channel records the current value, final value, step size, number of steps, number of cycles, minimum value, the last two parameters are used for effects. When using the API to set the color, it will sequentially change the final value, step size, and number of steps, and enable the gradient timer. The timer triggers a callback every 12 ms. The callback function will check the number of steps of each channel. As long as there are steps that have not been executed, it will add or subtract the current value according to the step size and update it to the underlying driver. When the number of steps of all channels is 0, it means that the gradient is completed, and the timer will be stopped at this time.

Low Power Implementation If the bulb is to pass power consumption certifications such as T20, after optimizing the lamp board power supply, some low-power configurations need to be made on the software side. In addition to the configurations mentioned in the Low Power Mode Usage Guide, some logic also needs to be done in the driver part. The bulb component has added related content in both the PWM and I2C dimming driver schemes. The specific logic is that when the switch is turned on, the I2C scheme calls the low-power command of the dimming chip itself to exit or enter low power. In the PWM scheme, the ESP32 needs to manage the power lock due to the use of the APB clock source, otherwise the light will flicker. When the light is on, take the power lock and disable dynamic frequency adjustment. When the light is off, release it. Other chips use the XTAL clock source and do not need to take any measures.

Maintain Color After Abnormal Restart The implementation in the bulb driver component is as follows: The PWM scheme can use the XTAL clock source, and the software restart does not affect the LEDC driver output; The I2C scheme needs to do software logic. After each successful operation of the lamp, the color needs to be recorded in the flash. After an abnormal restart, the color saved in the flash is used directly to light the lamp.

Power-on Lighting Speed When using the bulb driver component to light up, the time from the lamp board power supply to the lamp lighting is usually between $150 \sim 300$ ms. The main time consumption is in the log printing and various verifications related to chip security (Flash encryption, secure boot, etc.) in the ROM and bootloader stages. If encryption is not enabled and all logs at the startup stage are turned off, the lighting time will be less than 100 ms.

2.4 Recommended Tools

In this section, we will introduce some common tools. Although they are not provided by ESP, they play an important role in ESP development and debugging. These tools include but are not limited to Wireshark, Postman, etc. By

mastering the use of these tools, you will be able to carry out ESP development and debugging more efficiently, thereby saving valuable time and improving development efficiency.

2.4.1 Wireshark Packet Capture Tutorial on Windows

Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

This article provides a detailed tutorial on how to use Wireshark to capture Wi-Fi wireless packets on Windows 10.

Installing Wireshark

About Wireshark Wireshark (formerly Ethereal) is a network packet analysis tool. This tool is mainly used to capture network packets, automatically parse them, and display detailed information about the packets for user analysis. It can run on both Windows and Linux operating systems. This tool can be used to capture and analyze various protocol packets. This article will explain how to install and use this tool.

Download and Installation The Kali Linux system comes with the Wireshark tool, but it is not installed by default in the Windows system. Go to the Wireshark official website. Click on Get started to go to the download page.



In the Stable Release section, you can see that the latest version of Wireshark is 4.2.4, and download links for Windows (32-bit and 64-bit), Mac OS, and source code packages are provided. You can download the appropriate software package according to your operating system.

Here we download the 64-bit Windows installation package. Select Windows Installer(64-bit) to download. The downloaded file name is Wireshark-win64-4.2.4.exe. Double-click the downloaded software package to install it. During installation, use the default values and click the Next button. Note that Wireshark will ask the user whether to install the Npcap plugin synchronously (selected by default).

Npcap is an advanced packet capture and network sniffing software tool created by the Nmap project. It is a lightweight but powerful platform that allows users to perform real-time network traffic analysis and monitoring on the Windows operating system. Pay attention to the version of Npcap during this step. The early version of Npcap-1.7.1 could not use cmd to open the network card's monitor mode, which has been fixed in version 1.76. For more information, refer to source. The latest downloaded Wireshark comes with version 1.78, which can be installed directly.

Attention: Npcap will pop up a second installation page, where you need to check the two options in the figure below (**not checked by default**), otherwise you will not be able to capture 802.11 packets.

After installation, the Wireshark icon will appear in the Windows Start menu.







Packet Capture Network Card

The primary consideration is that the packet capture network card needs to support Monitor mode, followed by the wireless protocols and transmission rates supported by the network card.

Monitor Mode Monitor mode, or RFMON (Radio Frequency Monitor), refers to the working mode in which the wireless network card can receive all data streams passing through it, corresponding to other modes of IEEE 802.11 network cards, such as Master (router), Managed (ordinary mode network card), Ad-hoc, etc. Monitor mode does not distinguish the target MAC address of the received packets, which is similar to promiscuous mode. However, unlike promiscuous mode, monitor mode **does not require** a connection to be established with the wireless access point (AP) or Ad-hoc network. Monitor mode is a special mode unique to wireless network cards, while promiscuous mode applies to both wired and wireless network cards. Monitor mode is commonly used for network discovery, traffic monitoring, and packet analysis (source: Wikipedia).

Determine the Modes Supported by the Network Card You can use the Npcap tool to view the current mode of the network card, the supported modes, and turn on monitor mode.

1. Press win+R or enter cmd in the start menu to open the cmd command prompt window:



 $2. \ To view the GUID of the network card, enter netsh wlan show interfaces:$

D:\>netsh wlan show interfa	aces
系统上有 1 个接口:	
名称 描述 GUID 物理地址 状态 无线电状态 :	: WLAN 2 : Linksys WUSB600N Wireless-N USB Network Adapter with Dual-Band ver. 2 : 00:25:9c:e2:83:db : 己断开连接 硬件 开 软件 开
承载网络状态 : 未启动	

3. To view the current mode, copy the GUID, then enter WIanHelper.exe + GUID + modes, for example:

wlanhelper.exe 00971d40-8f3c-426d-bab3-5a38ce789c45 modes

You can then view the modes supported by the network card. Many built-in network cards in laptops will only have managed mode, in which case you need to purchase a wireless network card that supports monitor mode separately.



4. To turn on monitor mode for the network card, enter WlanHelper.exe + GUID + mode monitor, for example:

wlanhelper.exe 00971d40-8f3c-426d-bab3-5a38ce789c45 mode monitor

Success will be indicated by Success:

D:\>wlanhelper.exe 00971d40-8f3c-4 Success

Attention: cmd needs to be opened with administrator privileges, otherwise it will not be able to turn on monitor mode and will report an error. You can also enter mode to view the current mode. For more usage of WlanHelper, you can enter WlanHelper.exe -h to view.

Network Card Selection Choose a suitable wireless packet capture card according to your packet capture needs. If you need to capture Wi-Fi 6 packets, you need to choose a wireless card that supports the 802.11 ax protocol. Refer to the following table for the comparison of Wi-Fi protocol versions:

Generation	IEEE Standard	Radio Frequency (GHz)
Wi-Fi 6/6E	802.11ax	2.4, 5/6
Wi-Fi 5	802.11ac	5
Wi-Fi 4	802.11n	2.4, 5

You can search for keywords such as wireless packet capture and air packet capture card on shopping websites. Generally, cards with packet capture capabilities support monitor mode.



If the product does not provide detailed parameters, but provides the model of the wireless chip used, you can directly search for the model to view the detailed parameters of the chip:

This article takes the Cisco packet capture card with the Ralink RT3572 chip as an example. This chip supports the 802.11n protocol with a maximum rate of 300 Mbps and can capture empty packets of the Wi-Fi 4 protocol.

Specifications: FCC ID: RYK-WUBR507N Antenna: Built-in PCB Antenna, 2T2R Antenna Connector: U.FL IPEX / IPX Connector interface: USB 2.0/1.1, type A Wireless data rate: - 802.11b: 11, 5.5, 2, 1 Mbps - 802.11g: 54, 48, 36, 24, 18, 12, 9, 6Mbps - 802.11g: 54, 48, 36, 24, 18, 12, 9, 6Mbps - 802.11n: up to 300Mbps - 802. 11a: up to 300Mbps Frequency: 2.4GHz / 5GHz (NOT works with 802.11AC) IEEE WLAN Standard: IEEE 802.11 a/b/g/n Supported Systems: Kali Linux (Kali\ubuntu\Aircrack_ng), Linux, Windows XP/Vista/7/8/8.1/10 32/64-bit etc. Dimension (L x W x H): 9.50 x 4.00 x 1.60 cm / 3.74 x 1.58 x 0.63 inch Net Weight: 37.8g/0.08lb

What's in the box: 1x RT3572 USB WiFi Adapter

Configuring the Network Card

After getting the network card, you need to install the corresponding driver. Some network cards are plug-and-play and do not require installation. After plugging in the network card, open Control Panel-Device Manager, find the newly inserted network card in the Network Adapters column, double-click to view detailed information. If the driver is not compatible, there will be a yellow triangle:

- > 🖵 网络适配器
 - Intel(R) Ethernet Connection (17) I219-LM
 - 🚍 Linksys WUSB600N Wireless-N USB Network Adapter with Dual-Band ver. 2
 - WAN Miniport (IKEv2)
 - 🔄 WAN Miniport (IP)
 - WAN Miniport (IPv6)
 - WAN Miniport (L2TP)
 - WAN Miniport (Network Monitor)
 - WAN Miniport (PPPOE)
 - WAN Miniport (PPTP)
 - WAN Miniport (SSTP)
- > 🖿 系统设备

If your network card cannot be used after plugging in, this article introduces three methods to obtain the driver for the network card:

- Use the installation method provided by the seller when purchasing the network card, including but not limited to: the seller provides download links and installation methods, and the network card comes with a storage disk with the corresponding driver.
- Use third-party one-click installation software such as Driver Genius or Driver Master to automatically install the appropriate driver for the network card.
- Search for the model of the network card or the model of the wireless chip, and go to the corresponding chip official website or third-party software to directly download the corresponding driver.

After installation, some network cards need to restart the computer to adapt. After successful installation, open Network Adapter - WLAN in the lower right corner to search for nearby routers:



Capture Packets Using Wireshark

After installing Wireshark, you can run it to capture packets. Start Wireshark:

▲ Wireshark 网络分析器		_		×
文件(E) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(V) 无线(W) 工具(I) 帮助(H)				
📶 🗏 🕲 📘 🖹 🛛 🖸 🔍 🖛 \Rightarrow 🖀 🖉 💆 🚍 🔍 Q, Q, Q, II				
▲ 成用显示过滤器 < Ctrl-/>				+
Welcome to Wireshark				
捕获				
使用这个过滤器: 📕 输入捕获过滤器	显示所有接口▼			
以太网 4 ん	^			
Adapter for loopback traffic captureM				
本地连接* 6				
本地连接* 14				
本地连接*13				
今期2時で12 2011年12				
WLANZ				
Giscopping Giscopping				
Event Tracing for Windows (ETW) reader				
Random packet generator				
SSH remote capture				
UDP Listener remote capture	\sim			
学习				
User's Guide · Wiki · Questions and Answers · Mailing Lists · SharkFest · Wireshark Discord · Donate				
正在运行 Wireshark4.2.4 (v4.2.4-0-g1fe5bce8d665).接受自动更新。				
2 已准备好加载或捕获 无分组		配置	t: Defa	ult 📑

This is the main interface of Wireshark, which displays the currently available interfaces, such as Local Connection 8, WLAN 2, etc. To capture packets, you must first select an interface, indicating that the packets on this interface are captured. For basic concepts of network structure and protocols, you can refer to the blog: Understanding the Basic Principles of Wi-Fi Networks.

Capture Ethernet Packets You can use Wireshark to capture packets from your computer's network card and analyze the captured packets.

Attention: Loopback data of localhost accessed by the local machine does not go through the network card. To capture loopback data, you need to specify that the loopback data should be forwarded to the gateway first.

Wireshark can open and analyze packet capture files generated by other tools.

Select the Ethernet 4 interface to capture packets. Click on Start Capture at the top left corner or double-click on the interface name to begin capturing network data. While browsing the web on the local computer, Wireshark will capture data on the Local Connection interface.

-							
N) .	Time	Source	Destination	Protocol	Lengtl Primary Channel	Info
		74 9.247460	192.168.10.161	224.0.0.252	LLMNR	75	Standard query 0x6d83 AAAA WIN-7SOPNTUTP70
		75 9.249093	192.168.10.227	52.112.54.18	TLSv1.2	112	Application Data
L		76 9.290259	fe80::1c2f:9661:a818:1773	ff02::c	UDP	718	50063 → 3702 Len=656
		77 9.307201	192.168.10.86	239.255.255.250	UDP	698	50062 → 3702 Len=656
		78 9.354038	52.112.54.18	192.168.10.227	TLSv1.2	101	Application Data
		79 9.405657	192.168.10.227	52.112.54.18	TCP	54	52952 → 443 [ACK] Seq=59 Ack=48 Win=1025 Len=0
		80 9.457002	192.168.10.8	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
		81 9.498424	192.168.10.145	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
		82 9.590925	192.168.10.161	192.168.10.255	NBNS	92	Name query NB WIN-7SOPNTUTP70<00>
		83 9.624516	192.168.10.26	239.255.255.250	SSDP	218	M-SEARCH * HTTP/1.1
		84 9.840682	192.168.10.161	224.0.0.251	MDNS	81	Standard query 0x0000 AAAA WIN-7SOPNTUTP70.local, "QM" question
		85 9.840682	fe80::15b3:3814:ae20:91fa	ff02::fb	MDNS	101	Standard query 0x0000 AAAA WIN-7SOPNTUTP70.local, "QM" question
		86 9.840682	192.168.10.161	224.0.0.251	MDNS	81	Standard query 0x0000 A WIN-7SOPNTUTP70.local, "QM" question
		87 9.840682	fe80::15b3:3814:ae20:91fa	ff02::fb	MDNS	101	Standard query 0x0000 A WIN-7SOPNTUTP70.local, "QM" question
		88 9.939300	192.168.10.136	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
		89 10.042413	192.168.10.136	239.255.255.250	SSDP	218	M-SEARCH * HTTP/1.1
		90 10.132945	192.168.10.227	192.168.10.238	SSH	178	Client: Encrypted packet (len=124)
<							
~	Inte	ernet Protocol V	ersion 6, Src: fe80::1c2f:9	661:a818:1773, D	st: ff02:	:c	∧ 0030 00 00 00 00 00 0c c3 8f 0e 76 02 98 da a9 <mark>3c 3f</mark> ······
	6	0110 = Versi	ion: 6				0040 78 6d 6c 20 76 65 72 73 69 6f 6e 3d 22 31 2e 30 xml vers
	>	0000 0000	= T	raffic Class: 0x	00 (DSCP:	CS0, ECN: Not-ECT)	0050 22 20 65 66 63 67 64 69 66 67 3d 22 75 74 66 2d " encodi
		1110 1100 11	111 0000 1000 = Flow Label:	0xecf08			0070 70 65 20 78 6d 6c 6e 73 3a 73 6f 61 70 3d 22 68 pe xmlns
	F	Payload Length: 0	564				0080 74 74 70 3a 2f 2f 77 77 77 2e 77 33 2e 6f 72 67 ttp://ww
	Next Header: UDP (17)						0090 2f 32 30 30 33 2f 30 35 2f 73 6f 61 70 2d 65 6e /2003/05
	H	lop Limit: 1					00a0 76 65 6c 6f 70 65 22 20 78 6d 6c 6e 73 3a 77 73 velope"
	5	Source Address: 1	fe80::1c2f:9661:a818:1773				0000 61 30 22 68 74 74 70 3a 2f 2f 73 63 68 65 6d 61 a="http: 00c0 73 2e 78 6d 6c 73 6f 61 70 2e 6f 72 67 2f 77 72 c ymlega
	0	Destination Addre	ess: ff02::c				00d0 2f 32 30 30 34 2f 30 38 2f 61 64 64 72 65 73 73 /2004/08
\~	Use	r Datagram Proto	col, Src Port: 50063, Dst P	ort: 3702			00e0 69 6e 67 22 20 78 6d 6c 6e 73 3a 77 73 64 3d 22 ing" xml
	9	Source Port: 5000	53				00f0 68 74 74 70 3a 2f 2f 73 63 68 65 6d 61 73 2e 78 http://s

Wireshark will continue to capture data on the Local Connection. If you no longer need to capture, you can click on the Stop Capture button at the top left corner to stop capturing.



Capture Wireless Packets

Set Capture Options The menu bar Capture-Options (shortcut Ctrl + k) can enter the capture option setting interface:



Select the recently configured wireless network card. View the corresponding IP address by opening the drop-down menu:

	接口	流量	链路层	混杂	捕获长度(缓冲区 (M	监控模式	捕获过
<u> </u>	本地连接* 8		Ethernet	\checkmark	默认	2	_	
L	本地连接* 7		Ethernet		默认	2		
	本地连接* 6		Ethernet		默认	2	_	
	WLAN 2	_hm_hn_hn_	802.11 plus radiotap header		默认	2	\checkmark	
	地址: 169.254.161.106, fe80::7875:6402:aa	52:316d						
	本地连接* 14		Ethernet	\checkmark	默认	2		
	本地连接* 13		Ethernet	\checkmark	默认	2		
	本地连接* 12		Ethernet	\checkmark	默认	2		
	以太网 4	mm_mm_	Ethernet	\checkmark	默认	2	_	
	Adapter for loopback traffic capture	Manhaman	BSD loopback	\checkmark	默认	2	_	
۲	USBPcap1		USBPcap	_	_	_	_	
۲	Cisco remote capture		Remote capture dependent DLT			_		
۲	Event Tracing for Windows (ETW) reader		DLT ETW	_	_	_	_	
								>
王所	有接口上使用混杂模式					N	lanage li	nterface
		+_\\=					0	11

Attention: The IP address may change before and after connecting to the router. Please get the latest interface information through Capture-Refresh Interface List (shortcut F5):



Select Capture Card The network interface names configured by different computers may be different. Confirm the wireless network card for packet capture using the following methods:

Method 1 After connecting to a wireless network, use win+R or enter cmd in the start menu to open the cmd command prompt window:

最佳四	配	
	命令提示符 ^{系统}	

Enter ipconfig in the command bar and press enter, you can see the interface information of the current network, and the corresponding IP address can determine that the wireless network card is WLAN 2:

D:\>ipconfig	
Windows IP 配置	
以太网适配器 以太网 4:	
连接特定的 DNS 后缀	: : fe80::4994:7891:6d3d:72ae%4 : 192.168.10.227 : 255.255.255.0 : 192.168.10.1
无线局域网适配器 本地连接* 12:	
媒体状态	: 媒体已断开连接 :
无线局域网适配器 本地连接* 13:	
媒体状态	:媒体已断开连接 :
无线局域网适配器 WLAN 2:	
连接特定的 DNS 后缀	: : fe80::7875:6402:aa52:316d%2 : 169.254.161.106 : 255.255.0.0 :

Method 2 Alternatively, unplug the network card and try capturing with WLAN 2 in Wireshark. If the system prompts that the interface cannot be found, or Menu-Capture-Refresh Interface List, you can also determine the interface name of this network card.

🙍 Wire	eshark	×
8	The capture session could not be initiated on capture device "\Device\NPF_{00971D40-8F3C-426D-BAB3-5A38CE30C358}". (Error opening adapter: 找不到网络接口。 (2150891563)) Please check that you have the proper interface or pipe specified.	
	确定	

Method 3 Open the cmd command prompt window, enter netsh wlan show interfaces, and you can view the corresponding wireless network interface information.

Determine the Capture Channel Select the Wi-Fi channel you need to capture. If you want to capture the interaction information of the router under test, you can confirm the channel, bandwidth, and other information of the router as follows:

Method 1 Open the cmd command prompt window, connect the computer to the router under test, enter netsh wlan show interfaces, and you can view the channel of the current connection.

D:\>netsh wlan show interf	aces
系统上有 1 个接口:	
名称 描述 GUID 物理地址 状态 SSID BSSID BSSID 网络类型 无线电类型 身份验证 密码装模式 信道 接收速率(Mbps) 信号 配置文件	<pre>: WLAN 2 : Linksys WUSB600N Wireless-N USB Network Adapter with Dual-Band ver. 2 : 00971d40-8f3c-426d : 00:25:9c:e2 : 已连接 : 70:3a:73:84 : 44 : 80% : 00:25:9c:e2 : 60 : 144 : 144 : 80% : 00:25:9c:e2 : 60 : 144 : 144 : 80% : 00:25:9c:e2 : 60 : 144 : 144 : 80% : 00:25:9c:e2 : 60 : 144 : 144 : 80% : 00:25:9c:e2 : 60 : 00:25:9c:e2</pre>
承载网络状态 : 未启动	

Method 2 Check in the router administrator settings interface, not expanded here.

Method 3 Download Wi-Fi sniffing applications on your phone, such as WiFi Magic Box, wifiman, WiFi Analyzer, inSSIDer, WirelessMon, etc., to view the surrounding AP information and channel analysis.

Set Monitor Channel Open the cmd command prompt window, follow the method introduced in the previous *Determine the Modes Supported by the Network Card* section to enable the monitor mode of the network card.

÷	找AP	<	11
TP-LINK	TP-1011 TO INCLODIES CO., Cl	80Mhz 5G	-20 dBm CH. 153
>		WiFi6 40Mhz	-23 dBm CH. 5
TP-LINK	TP-10117 - 1010 - 11ES CO.,	40Mhz	-28 dBm CH. 11
	Fitwa		-34 dBm CH. 10

Attention: After opening this mode, the connected network will be disconnected, which is a normal phenomenon, because the common network card mode is managed, and it can be changed back after the subsequent packet capture is completed.

To enable the monitor mode, enter WlanHelper.exe + GUID + mode monitor, for example:

wlanhelper.exe 00971d40-8f3c-426d-bab3-5a38ce789c45 mode monitor

Configure the channel to be monitored, enter WlanHelper.exe + GUID + channel [value], channel 60 for example:

wlanhelper.exe 00971d40-8f3c-426d-bab3-5a38ce789c45 channel 60

Configure Wireshark capture options In the input field, select the corresponding network card interface, do not check the promiscuous mode, and check the monitor mode.

In the output field, select the path to be saved and the format to be stored.

Click Start or double-click the interface name to capture the wireless network data packets.

Usage of Wireshark Filters

Wireshark has set up two filters: capture filter and display filter.

• Capture filter:

Used to set filter conditions **before starting capture**. After setting the filter conditions, the packet capture tool will only capture packets that match the conditions; using the capture filter can reduce the captured network packets, reduce the burden of packet capture software and storage space, and the final packet capture file is also smaller, which is a necessary skill to improve efficiency;

接口	流量	链路层	混杂	捕获长度 (缓冲区 (M	监控模词	捕获
本地连接* 8		Ethernet	\checkmark	默认	2	_	
本地连接* 7		Ethernet	\checkmark	默认	2	_	
本地连接* 6		Ethernet	\checkmark	默认	2	_	
本地连接* 13		Ethernet	\checkmark	默认	2		
本地连接* 12		Ethernet	\checkmark	默认	2		
WLAN 2		802.11 plus radiotap header		默认	2	\checkmark	
地址: 169.254.161.106, fe80::7875:6402:	aa52:316d						
以太网 4		Ethernet	\checkmark	默认	2	_	
Adapter for loopback traffic capture		BSD loopback	\checkmark	默认	2	_	
OSBPcap1		USBPcap	_	_	_	_	
Oisco remote capture		Remote capture dependent DLT	_	_	_	_	
Event Tracing for Windows (ETW) read	er	DLT_ETW		_	_	_	
Random packet generator		Generator dependent DLT	_	_	_	_	
		•					
						Aspage l	atorfa
土所有按口工使用混示模式					P	lanage i	iteria

Input 输出 选项					
抓取并保存到一个永久文件					
文件: D:/log/wireshark	文件: D:/log/wireshark/240506				
Output format:					
已经	100000	◆ 分组			
□已经	1	↓ KB ∨			
□已经	1	seconds <			
当前时刻是整数倍于	1	♦ hours ∨			

Espressif Systems

• Display filter:

Used to set filter conditions **after capturing data**. After setting the filter conditions, only packets that match the conditions will be displayed on the display page, which helps engineers analyze packets.



Image source

Basic Usage of Capture Filter After opening the Wireshark software, the input box shown in the figure is the place to enter the capture filter conditions:

☑ 在所有接口上使用混杂模式		Μ
Capture filter for selected interfaces: 📕 脑入捕获过滤器	•	
	开始	¢

Click on the small green tag in the above figure (or through the menu: Capture – Capture Filters) to open commonly used capture expressions:

Note: The colon : in the capture expression usually means explanatory, without actual meaning.

Syntax of Capture Filter Expressions Wireshark capture filter expressions follow libpcap syntax. Filter expressions consist of one or more primitives. Primitives are usually composed of an id (name or number) and one or more modifiers.

- 1. Filter expression = Primitive $1 + Primitive 2 + \cdots$
- 2. Primitive = $id + Modifier 1 + Modifier 2 + \cdots$

:	输入捕获过滤器
	保存此过滤器
	删除此过滤器
-	管理捕获过滤器
	Ethernet address 00:00:5e:00:53:00: ether host 00:00:5e:00:53:00
	Ethernet type 0x0806 (ARP): ether proto 0x0806
	No Broadcast and no Multicast: not broadcast and not multicast
	No ARP: not arp
	IPv4 only: ip
	IPv4 address 192.0.2.1: host 192.0.2.1
_	IPv6 only: ip6
С	IPv6 address 2001:db8::1: host 2001:db8::1
E!	TCP only: tcp
	UDP only: udp
	Non-DNS: not port 53
	TCP or UDP port 80 (HTTP): port 80
=	HTTP TCP port (80): tcp port http
	No ARP and no DNS: not arp and port not 53
J	Non-HTTP and non-SMTP to/from www.wireshark.org: not port 80 and not port 25 and host www.wire

F	🚄 Wireshark · 捕获过滤器	
	V	
	Filter Name	Filter Expression
:	Ethernet address 00:00:5e:00:53:00	ether host 00:00:5e:00:53:00
	Ethernet type 0x0806 (ARP)	ether proto 0x0806
	No Broadcast and no Multicast	not broadcast and not multicast
	No ARP	not arp
	IPv4 only	ip
4	IPv4 address 192.0.2.1	host 192.0.2.1
	IPv6 only	ip6
L	IPv6 address 2001:db8::1	host 2001:db8::1
L	TCP only	tcp
	UDP only	udp
	Non-DNS	not port 53
Ξ.	TCP or UDP port 80 (HTTP)	port 80
1	HTTP TCP port (80)	tcp port http
	No ARP and no DNS	not arp and port not 53
L	Non-HTTP and non-SMTP to/from www.wir	eshark.org not port 80 and not port 25 and host www.wireshark.org
L		

- 3. Primitives can be combined with logical connectors and parentheses (), including:
 - And: Can be represented by the symbol && or the word and
 - Or: Can be represented by the symbol || or the word or
 - Not: Can be represented by the symbol ! or the word not

For example, this expression captures only packets that pass through the gateway \sup and belong to FTP ports or data:

gateway snup and (port ftp or ftp-data)

This example captures no arp type packets:

not arp

This example captures only tcp or udp type packets:

tcp || udp

It's worth mentioning that this filter comes with syntax checking, and the green frame at the bottom indicates correct syntax.

Note: For more usage and examples of capture filters, please refer to the Wireshark official explanation capture filters wiki and capture filters Gitlab.

Basic Usage of Display Filters After entering the packet capture page, click on the tag in the figure below, or through the menu Analyze- display filters to open commonly used display filter expressions:

■ 应用显示过滤器 < Ctrl-/>		
保存此过滤器	rimary Channel	Info
删除此过滤器		Null fi
管理显示过滤器		Acknow]
筛选器按钮首选项		QoS Dat
		Acknow]
Ethernet address 00:00:5e:00:53:00: eth.addr == 00:00:5e:00:53:00		Action,
Ethernet type 0x0806 (ARP): eth.type == 0x0806		Acknow]
Ethernet broadcast: eth.addr == ff:ff:ff:ff:ff:ff		Action,
No ARP: not arp		Null fi
IPv4 only: ip		Acknow.
IPv4 address 192.0.2.1: ip.addr == 192.0.2.1	11	Beacon
IPv4 address isn't 192 0 2 1: in addr != 192 0 2 1	11	Probe f
	11	Probe F
IP volumy, ip vo	11	Beacon
1006 address 2001:db8::1: 1006.addr = = 2001:db8::1	11	Beacon
ICP only: tcp		Request
UDP only: udp		Null fi
Non-DNS port: !(udp.port == 53 tcp.port == 53)		
TCP or UDP port is 80 (HTTP): tcp.port == 80 udp.port == 80		
HTTP: http	ace \Device\M	IPF_{009
No ARP and no DNS: not arp and not dns		
Non-HTTP and non-SMTP to/from 192.0.2.1: ip.addr == 192.0.2.1 and tcp.port not in {80, 25}		

Syntax of Display Filter Expressions Similar to capture filter expressions, display filter expressions can also be seen as a combination of primitives. The difference is that the primitives of display filter expressions consist of option + option relationship + option value. For example, in tcp.port == 80, tcp. port is the option, == is the option relationship, and 80 is the option value. The entire expression means: only display packets with a tcp port number (including sending and receiving) of 80.



🧲 Wireshark · 显示过滤器

Fil	ter Name	Filter Expression
	Ethernet address 00:00:5e:00:53:00	eth.addr == 00:00:5e:00:53:00
	Ethernet type 0x0806 (ARP)	eth.type == 0x0806
	Ethernet broadcast	eth.addr == ff:ff:ff:ff:ff:ff
	No ARP	not arp
	IPv4 only	ip
	IPv4 address 192.0.2.1	ip.addr == 192.0.2.1
	IPv4 address isn't 192.0.2.1	ip.addr != 192.0.2.1
	IPv6 only	ipv6
	IPv6 address 2001:db8::1	ipv6.addr == 2001:db8::1
	TCP only	tcp
	UDP only	udp
	Non-DNS port	!(udp.port == 53 tcp.port == 53)
	TCP or UDP port is 80 (HTTP)	tcp.port == 80 udp.port == 80
	НТТР	http
	No ARP and no DNS	not arp and not dns
	Non-HTTP and non-SMTP to/from 192.0.2.1	ip.addr == 192.0.2.1 and tcp.port not in {80, 25}

A reminder once again, display filter expressions and capture filter expressions should not be confused. The form of a capture filter expression is: tcp port80. The three parts of the expression are explained as follows:

- Option: An option can be a protocol (such as tcp, udp, http, etc.), frame, and other objects. Refer to existing examples or directly input the protocol name to see if there are any prompts.
- Option relationship: Used to define the relationship between the option and the option value. Common option relationships are as follows:

English	Alias	C- like	Description	Example
eq	any_eq	==	Equal (any if more than one)	ip.src == 10.0.0.5
ne	all_ne	!=	Not equal (all if more than one)	ip.src != 10.0.0.5
	all_eq	===	Equal (all if more than one)	ip.src === 10.0.0.5
	any_ne	!==	Not equal (any if more than one)	ip.src !== 10.0.0.5
gt		>	Greater than	frame.len > 10
lt		<	Less than	frame.len < 128
ge		>=	Greater than or equal to	frame.len ge 0x100
le		<=	Less than or equal to	frame.len <= 0x20
contains			Protocol, field or slice contains a value	sip.To contains "a1762"
matches		~	Protocol or text field matches a Perl-compatible regular expression	http.host matches "acme\\. (org com net)"

Table 6.6. Display Filter comparison operators

Table source: Wireshark Official Website

• Option value: The option value can be a number, such as a decimal number 1500, a hexadecimal number 0x5dc, a binary number 0b10111011100, a boolean value (where 1 represents true, 0 represents false), a MAC address number (such as eth.dst == ff:ff:ff:ff:ff:ff), an IP address (such as ip.addr == 192.168.0.1), a string (such as http.request.uri == "https://www.wireshark.org/"), a time (such as ntp.xmt ge "2020-07-0412:34:56"), etc.

Multiple expressions can also be connected by logical connectors to form advanced display filter expressions.

Table source: Wireshark Official Website

Here, [...] represents the substring selector, such as eth.src[0:3] == 00:00:83 indicates that the data of the 3 bytes starting from offset address 0 is 00:00:83. The in represents the member selector, commonly used to form a set of option values. For example, tcp.port in {80,441,8081} means to filter the tcp packets with ports 80, 441, 8081. It is equivalent to tcp.port == 80 || tcp.port == 441 || tcp.port == 8081.

Note: For more usage methods and examples of capture filters, please refer to the Wireshark official explanation

English	C-like	Description	Example
and	&&	Logical AND	ip.src==10.0.0.5 and tcp.flags.fin
or	П	Logical OR	ip.src==10.0.0.5 or ip.src==192.1.1.1
xor	^^	Logical XOR	tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29
not	!	Logical NOT	not llc
[]		Subsequence	See "Slice Operator" below.
in		Set Membership	http.request.method in {"HEAD", "GET"}. See "Membership Operator" below.

Table 6.7. Display Filter Logical Operations

capture filters wiki and capture filters Gitlab.

Packet Analysis

After mastering the packet capture method of Wireshark, you can perform preliminary analysis on the captured packets. Here are some excellent packet analysis examples:

- Wireshark Packet Capture Analysis of WLAN Connection Process
- Using Wireshark to Analyze the Specific Process of Ping Communication
- Wireshark Packet Capture Analysis of TCP Three-Way Handshake

At the end of this article, we will share some tips on using Wireshark in packet analysis:

Tip 1 In addition to manually entering display filter expressions in the input box, you can also select an option from the packet capture data, right-click and choose Apply as Filter, then select Selected, Not Selected, or the more advanced and or logical operations according to your needs. The following figure filters the RTS packets with subtype as 0x001b, which is equivalent to entering wlan.fc.type_subtype == 0x001b in the display filter bar.

Tip 2 Use shaders to assist in analyzing various packets. Select an option from the packet capture data, right-click and choose Apply as Filter with Coloring to color packets that meet specific conditions. The following figure colors packets with a Receiver address of Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) as color 3:

You can manage or reset existing rules in View - Conversation Coloring:

Tip 3 You can drag any field of interest to the list bar for display. For example, the following figure drags the current channel field to the Packet List Pane, making it easy to view the channel information for each packet:

Right-click the list bar to manage the settings rules for each column:

You can also choose Edit column to set:

		<u> </u>				
wlan.fc.type_subtype =	= 0x001b					
No. Time	Source	Destination	Protocol	Lengtl Primary Cha	annel Info	
8 0.019186	Espressif_4d:d4:ac (68:b	TpLinkTechno_8	802.11	35	Request-to-s	end, Flags=C
15 0.024109	56:47:29:7b:52:a6 (56:47	06:69:6c:b9:2a	802.11	35	Request-to-s	end, Flags=C
34 0.122982	Espressif_4d:d4:ac (68:b	TpLinkTechno_8	802.11	35	Request-to-s	end, Flags=C
72 0.241814	Espressif_4d:d4:ac (68:b	TpLinkTechno_8	802.11	35	Request-to-s	end, Flags=C
104 0.296168	6:69:6c:b9:2c:3c (66:69	06:69:60:09:2a	802.11	35	Request-to-s	end, Flags=C
138 0.366866	06:69:6c:b9:2a:3a (06:69	56:47:29:7b:52	802.11	35	Request-to-s	end, Flags=C
144 0.392024	56:47:29:7b:52:a6 (56:47	06:69:6c:b9:2a	802.11	35	Request-to-s	end, Flags=C
163 0.440814	56:47:29:7b:52:64 (56:47	06:69:6c:b9:2a	802.11	35	Request-to-s	end, Flags=C
166 0.446242	Espressif_4d:d4:ac (68:b…	TpLinkTechno_8	802.11	35	Request-to-s	end, Flags=C
218 0.661299	Espressif_4d:d4:ac (68:b…	TpLinkTechno_8…	802.11	35	Request-to-s	end, Flags=C
223 0.662941	Espressif_4d:d4:ac (68:b	TpLinkTechno_8	802.11	35	Request-to-s	end, Flags=C
259 0.761760	Espressif_4d:c Expand S	ubtrees		35	Request-to-s	end, Flags=C
277 0.826436	Espress1t_4d:c 折叠子树			35	Request-to-s	end, Flags=C
317 0 904350	06·69·6c·b9·22 全部展开			35	Request-to-s	end Flags=
321 0.911095	06:69:6c:b9:2a 全部折叠			35	Request-to-s	end, Flags=C
323 0.918433	Espressif 4d:c			35	Request-to-s	end. Flags=
<	应用为列	C	trl+Shift+I			
Frame Length: 35	bytes (280 bit 作为过滤	器应用		作为过滤器应用	围: wlan.fc.type_subtyp	e == 0x001b
Capture Length:	35 bytes (280 b 准备作为i	寸滤器			71 <u>-</u> 71	d9 b3 80 80
[Frame is marked	: False] 」 対话过滤	¥2		选中		
[Frame is ignore	a: Faise」 http://aisean.u/ 用过滤器	皆色		非选中		
✓ Radiotap Header v0.	Length 15 追踪流			…且选中		
Header revision:	0			或选中		
Header pad: 0	复制			▶ …且不选中		
Header length: 1	.5 显示分组:	字节 C	trl+Shift+O	或不选中		
✓ Present flags	导出分组	字节流(B) C	trl+Shift+X			
> Present flags	word: 0x000000			-		
> Flags: 0x50	Wiki 协议	页面				
Channel frequence	y: 0 过滤器字	没参考				
Δntenna signal:	0 dBm 协议首选J	页				
✓ 802.11 radio inform	nation 解码为(A)		trl+Shift+U			
Signal strength	(dBm): 0 dBm 柱空链控	か分组(I)				
✓ IEEE 802.11 Request	-to-send, Flags	5月元(5)				
Type/Subtype: Re	quest-to-send (×	
<					>	
🛛 🕘 🌋 🛛 Type and subtyp	e combined (as bytes (for Contro	l Frame Extension sul	otypes) or n	ibbles) (wlan.fc.type_	subtype), 1 byte(s)	分
9 0.019260	5	Espressit_4d	:d 802.1	1 29	Clear-	to-send, Flags=
10 0.019336	Espressit_4d:d4:ac	IpLinklechno	_ ² Expa	and Subtrees		nction (No data), SN=3
11 0.019387	Francis Advida	Espressit_4d	: 折叠	子树		edgement, Flags=
12 0.022399	Espressif_4d:d4:ac	Thurkiechno	・ 全部	展开		nction (No data), SN=30
13 0.023693	Espressit_4d:d4:ac	IpLinklechno	1 全部	折叠		nction (No data), SN=5
14 0.023/38	E6.47.20.7b.E2.a6 (E6.4)	Espressit_4d				edgement, Flags=
15 0.024109	Neu H2CTachea 28: 60:20	Proodcost	~ 应用	为列	Ctrl+Shift+I	from SN-1055 SN-0
17 0.020391	HuaweiTechno e9.7e.fo	RasphennyPiE	4 化巨大	讨滤器应田	•	esponse. SN=568 EN-0
18 0.046569	XiaomiMobile 88.11.56	Broadcast				frame, SN=2164 EN-0
111 17.1/40 10 7	A.I.A.M.IPRAZITE 00.11.30	or constraint	/准备	1F/J12)滤器	•	
Header revision: 0				过滤器	•	12a
Header pad: 0			用过	滤器看色	•	1 颜色1 b6
Header length: 15			追踪	流	•	2 颜色 2
✓ Present flags			信// ···			3 颜色 3
> Present flags word: 0x0000002a			反制			4 颜色 4
> Flags: 0x50			显示	分组字节	Ctrl+Shift+O	
Channel frequency: 0			导出	分组字节流(B)	Ctrl+Shift+X	5 颜色 5
> Channel flags: 0x0000						6 颜色 6
Antenna signal: 0 dBm			Wiki	协议页面		7 颜色 7
000 44 11 1 6			>+>.=	照今61.关于/		

802.11 radio information 过滤器字段参考 8 颜色 Signal strength (dBm): 0 dBm b\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	Antenna signal: 0 dBm	WIKI WIX 风国		7	颜色 7	
Signal strength (dBm): 0 dBm 协议首选项 9 颜色 IEEE 802.11 Acknowledgement, Flags:C 解码为(A) Ctrl+Shift+U 10 颜色 Type/Subtype: Acknowledgement (0x001d) Frame Control Field: 0xd400 好码为(A) Ctrl+Shift+U 10 颜色 .000 0000 0000 0000 = Duration: 0 microseconds 在新窗口中显示已链接的分组 新建 Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) 在新窗口中显示已链接的分组 新建	802.11 radio information	过滤器字段参考		8	新缶 8	
IEEE 802.11 Acknowledgement, Flags:C 解码为(A) 9 颜色 Type/Subtype: Acknowledgement (0x001d) 解码为(A) Ctrl+Shift+U > Frame Control Field: 0xd400 转至链接的分组(L) .000 0000 0000 0000 = Duration: 0 microseconds 在新窗口中显示已链接的分组 Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) 在新窗口中显示已链接的分组	Signal strength (dBm): 0 dBm	协议首选项	•		BRE U	
Type/Subtype: Acknowledgement (0x001d) 解码为(A) Ctrl+Shift+U 10 颜色 > Frame Control Field: 0xd400 转至链接的分组(L) 新建 .000 0000 0000 0000 = Duration: 0 microseconds 在新窗口中显示已链接的分组 新建 Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) 在新窗口中显示已链接的分组 新建	IEEE 802.11 Acknowledgement, Flags:C			9	颜色 9	
> Frame Control Field: 0xd400 .000 0000 0000 = Duration: 0 microseconds Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) Frame check sequence: 0x20000000 [unverified]	Type/Subtype: Acknowledgement (0x001d)	解码为(A)	Ctrl+Shift+U	10	颜色 10	
.000 0000 0000 = Duration: 0 microseconds 在新窗口中显示已链接的分组 Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) Frame check sequence: 0x20000000 [unverified]	> Frame Control Field: 0xd400	转至链接的分组(L)			新建着色规则	
Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac) Frame check sequence: 0x20000000 [unverified]	.000 0000 0000 0000 = Duration: 0 microseconds	在新窗口中显示已链接的分组		-		
Frame check sequence: 0x20000000 [unverified]	Receiver address: Espressif_4d:d4:ac (68:b6:b3:4d:d4:ac)	L		-		
	Frame check sequence: 0x20000000 [unverified]					
	First and the second					
1	/ 0.015100	00.05.0C.05.28.38 (00.05.	30.41.23.10.32	002.11	47	OUZ.II DIULK MLK, FIEBS
----	-------------	---------------------------	----------------	--------	-----	---
1	8 0.019186	Espressif_4d:d4:ac (68:b	TpLinkTechno_8	802.11	35	Request-to-send, Flags=C
	9 0.019260		Espressif_4d:d	802.11	29	Clear-to-send, Flags=C
I	10 0.019336	Espressif_4d:d4:ac	TpLinkTechno_8	802.11	43	Null function (No data), SN=3005, FN=0, Flags=RTC
	11 0.019387		Espressif_4d:d	802.11	29	Acknowledgement, Flags=C
I	12 0.022399	Espressif_4d:d4:ac	TpLinkTechno_8	802.11	43	Null function (No data), SN=3007, FN=0, Flags=TC
L	13 0.023693	Espressif_4d:d4:ac	TpLinkTechno_8	802.11	43	Null function (No data), SN=3008, FN=0, Flags=TC
1	14 0.023738		Espressif_4d:d	802.11	29	Acknowledgement, Flags=C
I	15 0.024109	56:47:29:7b:52:a6 (56:47	06:69:6c:b9:2a	802.11	35	Request-to-send, Flags=C
I.	16 0.026391	NewH3CTechno 28:f0:2e	Broadcast	802.11	240	Beacon frame. SN=1055. FN=0. Flags=C. BI=100. SSID="Lenovo"[M

Tip 4 Sometimes we need to debug complex issues and capture multiple logs from different devices. Setting absolute timestamps can help locate complex issues across devices. Open the column settings bar using the method in Tip 3, select Absolute date, and you can display the absolute time for each packet:

After setting:

Chinese Reference Documents

- Network Packet Capture Tool Wireshark Download Installation & Detailed Tutorial
- Basic Use of Wireshark Enabling Packet Capture and Filtering
- Basic Use of Wireshark Filtering and Viewing Captured Data
- Wi-Fi Packet Capture Tutorial on Windows

视图	퇸(V)	跳转(G)	捕获(C)	分析(A)	统计(S)	电话((Y) 无	銭(W)	ΤĦ	Į(T) ≉	帮助(H
\checkmark	±Τ	具栏(M)				10		9 8			
\checkmark	过滤	器工具栏(F)									
\checkmark	状态	≚(S)				H		D .			- ·
						_ on		Protoc		Lengti	Prim
	全屏(F)		F11			E . 34	802.1	1	3516	
\checkmark	分组初	列表(L)				2:0	c·11	802.1	1	920	
~	分组ì	主(D)				ech	no 8	802.1	1	43	
	分组	之节流(B)				9:7	b:52	802.1	1	47	
	分组	외(D)				ech	no_8	802.1	1	35	
1		SI(U)				_if_	4d:d	802.1	1	29	
	时间	显示格式(T)			•	' ech	no_8	802.1	1	43	
	名称的	解析(U)			•	· if_	4d:d	802.1	1	29	
	编动	7)				ech	no_8	802.1	1	43	
	SHUX(,2)			•	ech	no_8	802.1	1	43	
	展开	子树(X)		Shift	+右方向	1T_	4a:a	802.1	1	29	
	折叠	子树		Shift	+左方向	5+	9.2a	802.1	1 1	240	
	展开会	全部(E)		Ctrl+	-右方向	-vP	iF 4	802.1	1	544	
	收起	全部(A)		Ctrl+	+左方向	st	_	802.1	1	404	
	* 7					ryP	iF_4	802.1	1	256	
	看巴	分组列表				rvP	iF 4	802.1	1	256	
	着色热	现则(C)									_
	对话	着色			•	1	颜色 1		C	trl+1	
1	重置	布局		Ctrl+	+Shift+W	2	颜色 2		C	trl+2	
	调整	列宽		Ctrl+	+Shift+R	3	颜色 3		С	trl+3	
						4	颜色 4		С	trl+4	
	内部				,	5	颜色 5		С	trl+5	
	在新聞	窗口显示分错	组(W)			6	颜色 6		С	trl+6	
	重新	載入为文件	格式/捕获	Ctrl+	+Shift+F	7	颜色 7		с	trl+7	
3	重新	加载(R)		Ctrl+	⊦R	8	颜色 8		С	trl+8	
reng	th (dBm): 0 a	dBm				統合 0		- -	trl±0	
Requ	Request-to-send, Flags:C							0	C	1179	
ype: Request-to-send (0x001b)											
trol Field: 0xb400							重置着	色	С	trl+空格	
011	1 10	10 = Dura	ation: 1		新建着色规则						
address: 06:69:6c:b9:2a:3a (06:69:6c:b9:2											

```
v... ... = Keserved: 0
v Tagged parameters (429 bytes)
> Tag: SSID parameter set: "Audio_CI"
> Tag: Supported Rates 1(B), 2(B), 5.5(B), 11(B), 6,
v Tag: DS Parameter set: Current Channel: 11
Tag Number: DS Parameter set (3)
Tag length: 1
Current Channel: 11
> Tag: Country Information: Country Code CN, Environm
```

> Tag: FRP Information

No.	Time	Source	Destination	Protocol	Lengtł	Current Channel	Int	fo		
	12 0.022399	Espressif_4d:d4:ac	TpLinkTechno_8	802.11	43			Align Left		· · · ·
	13 0.023693	Espressif_4d:d4:ac	TpLinkTechno_8	802.11	43			Align Center		
	14 0.023738		Espressif_4d:d	802.11	29			Align Right		
	15 0.024109	56:47:29:7b:52:a6 (56:47	06:69:6c:b9:2a	802.11	35					
	16 0.026391	NewH3CTechno_28:f0:2e	Broadcast	802.11	240	11	L	列首选项		100
	17 0.042932	HuaweiTechno_e9:7e:f0	RaspberryPiF_4	802.11	544	11	L	Edit Column		=10
	18 0.046569	XiaomiMobile_88:11:56	Broadcast	802.11	404	11	L	Resize to Contents		100
	19 0.048856	UTTTechnolog_ab:db:70	RaspberryPiF_4	802.11	256	11	L	Resize Column to Width		I=2
	20 0.051035	UTTTechnolog_ab:db:70	RaspberryPiF_4	802.11	256	11	L	Recolve Names		I=2
	21 0.055514	XiaomiMobile_b9:6b:be	RaspberryPiF_4	802.11	523	11	L	Resolve Maines		=10
	22 0.069932	HuaweiTechno_e9:7e:f1	Broadcast	802.11	258	11	~	No.	Number	.00,
	23 0.073936	XiaomiMobile_88:11:56	96:32:53:84:d8	802.11	484	11	~	Time	Time (format as specified)	I=1
	24 0.086600	XiaomiMobile_b9:6b:be	RaspberryPiF_4	802.11	523	11		Source	Source address	=10
	25 0.091335	XiaomiMobile_88:11:56	96:32:53:84:d8	802.11	484	11	Ľ,	Destination	Destination address	I=1
	26 0.092000		00:69:6c:b9:2a	802.11	29		~	Destination	Destination address	
	27 0.094568	TpLinkTechno_6b:d6:c2	Broadcast	802.11	247	11	~	Protocol	Protocol	.00,
	28 0.095877	53:cf:f5:55:c5:2b (53:cf	la:a3:df:5e:ea…	802.11	76		\checkmark	Length	Packet length (bytes)	
	29 0.104560	TnlinkTechno 80:8c:81	Broadcast	802.11	298	11	~	Current Channel	wlan.ds.current_channel	100
<							~	Info	Information	
	0	= EPD: Not Implemen	ted							43
	.0	= Reserved: 0		Remove this Column		14				

标题: Info		类型: Information		~ 字	段: 输入字段	发生:	Resolve Names: 🗹	确定
No.	Time	Source	Destination	Protocol	Lengtl Current Channel Info			

标题: Time Y型: Time (format as specified) ✓ No. Time Source Espressif 12 0.022399 Espressif 13 0.023693 Espressif 14 0.023738 56:47:29: 16 0.026391 NewH3CTec 17 0.042932 HuaweiTec 18 0.046569 XiaomiMob 19 0.048856 UTTTechno 20 0.051035 UTTTechno 21 0.055514 XiaomiMobile_b9:6b:be RaspberryPiF_4 802.11							
No.TimeSourceAbsolute date, as YYYY-MM-DD, and timeocol120.022399EspressifAbsolute date, as YYYY/DOY, and time11130.023693EspressifAbsolute date, as YYYY/DOY, and time11140.023738Cumulative Bytes11150.02410956:47:29:Custom11160.026391NewH3CTecDelta time11170.042932HuaweiTecDelta time displayed11190.048856UTTTechnoDest addr (unresolved)11200.051035UTTTechnoDest port (resolved)11210.055514XiaomiMobile_b9:6b:beRaspberryPiF_4802.11	标题:	Time		类型:	Time (format as specified)	\sim	字
12 0.022399 Espressif 13 0.023693 Espressif 14 0.023738 Espressif 15 0.024109 56:47:29: 16 0.026391 NewH3CTec 17 0.042932 HuaweiTec 18 0.046569 XiaomiMob 19 0.048856 UTTTechno 20 0.051035 UTTTechno 21 0.055514 XiaomiMobile_b9:6b:be RaspberryPiF_4	No.		Time	Source	Absolute date, as YYYY-MM-DD, and time	^	ocol
14 0.023738 Cumulative Bytes 11 15 0.024109 56:47:29: Custom 11 16 0.026391 NewH3CTec Delta time 11 17 0.042932 HuaweiTec Delta time displayed 11 18 0.046569 XiaomiMob Dest addr (resolved) 11 19 0.048856 UTTTechno Dest port (resolved) 11 20 0.051035 UTTTechno Dest port (resolved) 11 21 0.055514 XiaomiMobile_b9:6b:be RaspberryPiF_4 802.11		12 13	0.022399 0.023693	Espressif Espressif	Absolute date, as YYYY/DOY, and time Absolute time		11 11
16 0.026391 NewH3CTec Delta time 11 17 0.042932 HuaweiTec Delta time displayed 11 18 0.046569 XiaomiMob Dest addr (resolved) 11 19 0.048856 UTTTechno Dest port (resolved) 11 20 0.051035 UTTTechno Dest port (resolved) 11 21 0.055514 XiaomiMobile_b9:6b:be RaspberryPiF_4 802.11		14 15	0.023738 0.024109	56:47:29:	Cumulative Bytes Custom		11 11
180.046569XiaomiMobDest addr (resolved)11190.048856UTTTechnoDest addr (unresolved)11200.051035UTTTechnoDest port (resolved)11210.055514XiaomiMobile_b9:6b:beRaspberryPiF_4802.11		16 17	0.026391	NewH3CTec HuaweiTec	Delta time Delta time displayed		11
20 0.051035 UTTTechno Dest port (resolved) 11 21 0.055514 XiaomiMobile_b9:6b:be RaspberryPiF_4 802.11		18 19	0.046569	XiaomiMob UTTTechno	Dest addr (resolved) Dest addr (unresolved)		11
		20 21	0.051035 0.055514	UTTTechno XiaomiMob	Dest port (resolved) 	✓	11 11

	Time		Sc
12	2024-05-08	11:56:45.093679	Es
13	2024-05-08	11:56:45.094973	Es
14	2024-05-08	11:56:45.095018	
15	2024-05-08	11:56:45.095389	56
16	2024-05-08	11:56:45.097671	Ne
17	2024-05-08	11:56:45.114212	Hι
18	2024-05-08	11:56:45.117849	Xi
19	2024-05-08	11:56:45.120136	UT
20	2024-05-08	11:56:45.122315	UT
21	2024-05-08	11:56:45.126794	Xi
22	2024-05-08	11:56:45.141212	Hι
23	2024-05-08	11:56:45.145216	Xi
24	2024-05-08	11:56:45.157880	Xi
25	2024-05-08	11:56:45.162615	Xi
26	2024-05-08	11:56:45.163280	
27	2024-05-08	11:56:45.165848	Τŗ
28	2024-05-08	11:56:45.167157	53
			_