



ESP-Techpedia_EN



Release master
Espressif Systems
Jun 26, 2024




Table of contents



Table of contents	i
1 ESP-FAQ	3
2 ESP Friends	5
2.1 Get Started	5
2.1.1 Board selection	5
2.1.2 Flash Programming	20
2.1.3 Environment Setup	37
2.1.4 Getting Started with Project Development	42
2.2 Advanced Development	45
2.2.1 Component Management and Usage	45
2.2.2 Advanced Code Debugging	49
2.2.3 Performance Optimization	88
2.2.4 Template	89
2.3 Solution Introduction	90
2.3.1 Template	90
2.4 Recommended Tools	90
2.4.1 Wireshark Packet Capture Tutorial on Windows	90

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.

	
ESP Friends	ESP FAQ

Chapter 1

ESP-FAQ

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.

Table 1: Chip Comparison

Chip	Year of Release	Application Scenarios	Wireless Functions	GPIO	SRAM	PSRAM Support
ESP32-C6	2023	Ultra-low power IoT devices with long battery life, Thread border routers, Matter gateways, Zigbee bridges	BLE 5.0 + Wi-Fi 6 + Thread + Zigbee	23	512 KB	×
ESP32-C2	2022	Sockets, lighting, sensors, simple smart home appliances	BLE 5.0 + Wi-Fi 4	14	272 KB	×
ESP32-H2	2021	Thread border routers, Matter gateways, Zigbee bridges	BLE 5.0 + Thread + Zigbee	19	230 KB	✓
ESP32-S3	2020	Smart cameras, face recognition, voice recognition, voice wake-up, real-time data collection and processing, complex peripheral control	BLE 5.0 + Wi-Fi 4	36	320 KB	✓
ESP32-S2	2020	Real-time data collection and processing, complex peripheral control	Wi-Fi 4	36	320 KB	✓
ESP32-C3	2020	Electrical lighting, switch sockets, smart home appliances, industrial control field	BLE 5.0 + Wi-Fi 4	15	400 KB	×
ESP32	2016	Recommended to use the latest released ESP32-S3	BLE 4.2 + BT + Wi-Fi 4	26	520 KB	✓
ESP8266	2014	Recommended to use the latest released ESP32-C2 or ESP32-C3 . ESP8266 is about to reach the 12-year supply guarantee time	Wi-Fi 4	11	160 KB	×

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 chips that integrates chips, crystals, antennas, and flash. Espressif's modules typically include pre-integrated wireless functions such as Wi-Fi, Bluetooth, and are certified with FCC, CE, and other regulatory certifications. This allows developers to focus more on applications development without the need to manage the complexities of wireless communication, thereby accelerating time-to-market for products.
- 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 developers' 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** can usually accelerate the product's time to market, as they have pre-integrated wireless functions (such as Wi-Fi, Bluetooth) and relevant certifications. Developers can focus more on application development without dealing with the details of wireless communication

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

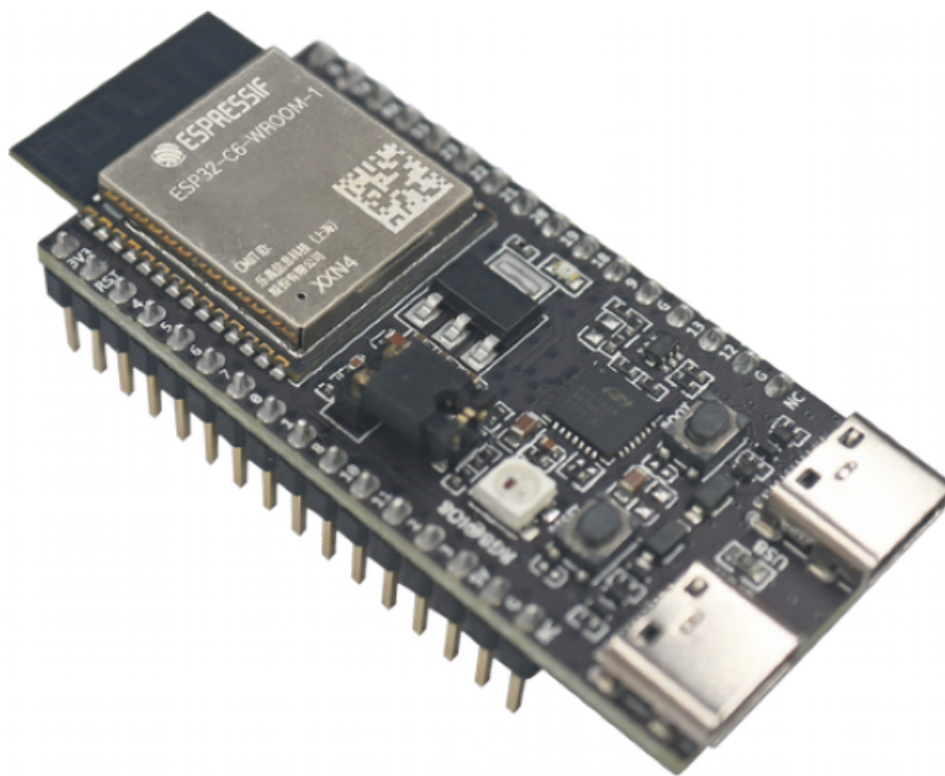
ESP32-C6

Supported Features:

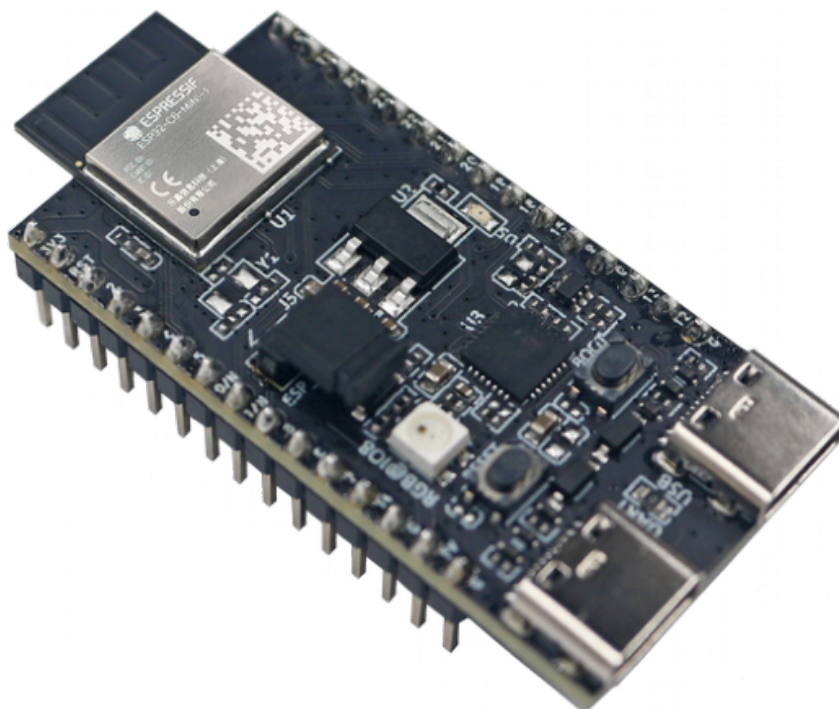
- 30 (QFN40) or 22 (QFN32) programmable GPIO pins, supporting SPI, UART, I2C, I2S, RMT, TWAI, and PWM
- Can be used for development solutions: ultra-low power IoT devices with long battery life, Thread border routers, Matter gateways, Zigbee bridges

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](#)

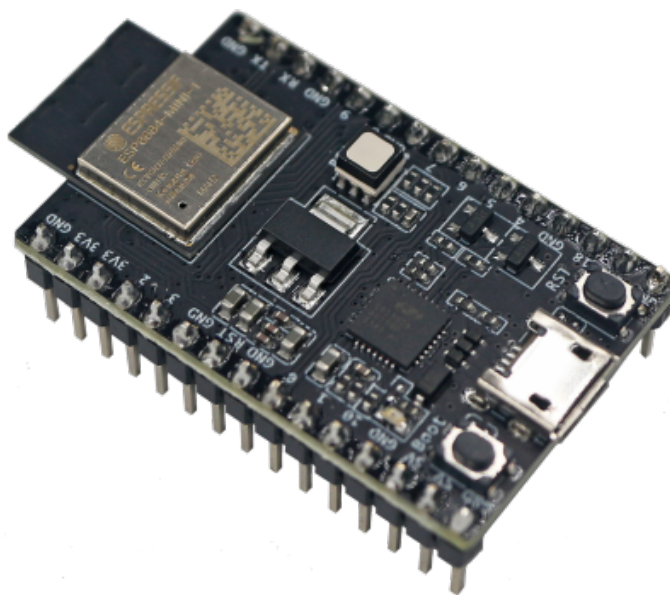
ESP32-C2

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](#)

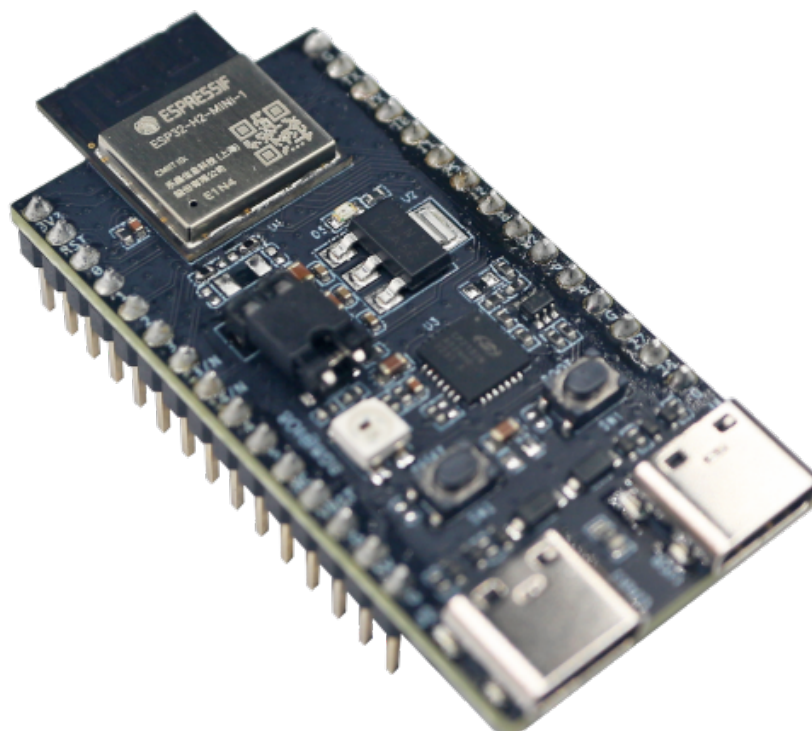
ESP32-H2

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](#)

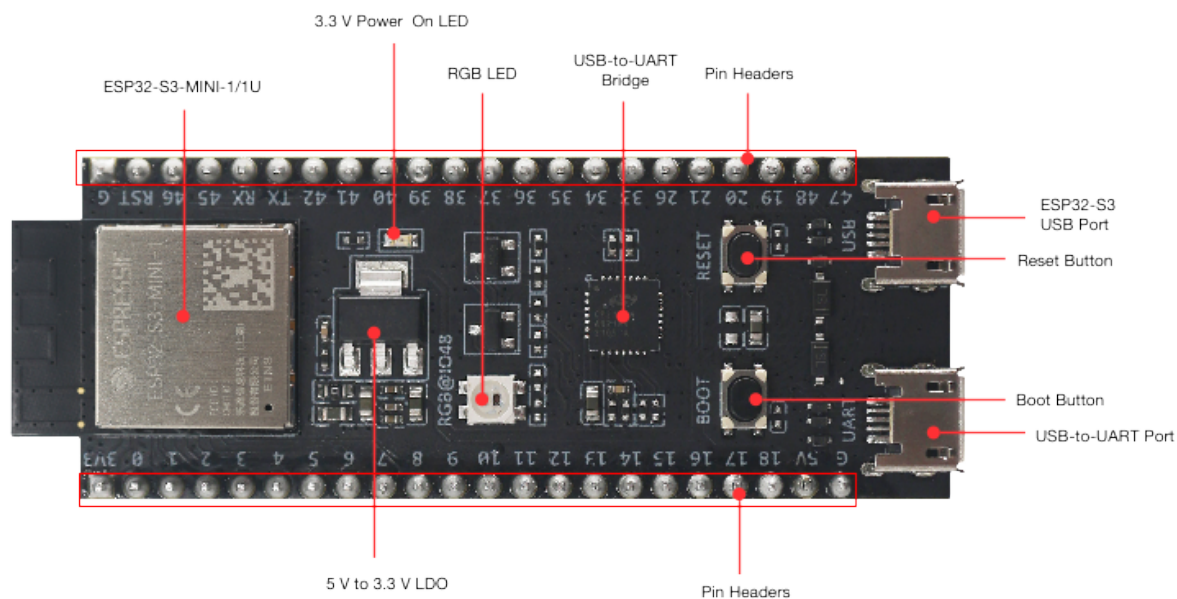
ESP32-S3

Supported Features:

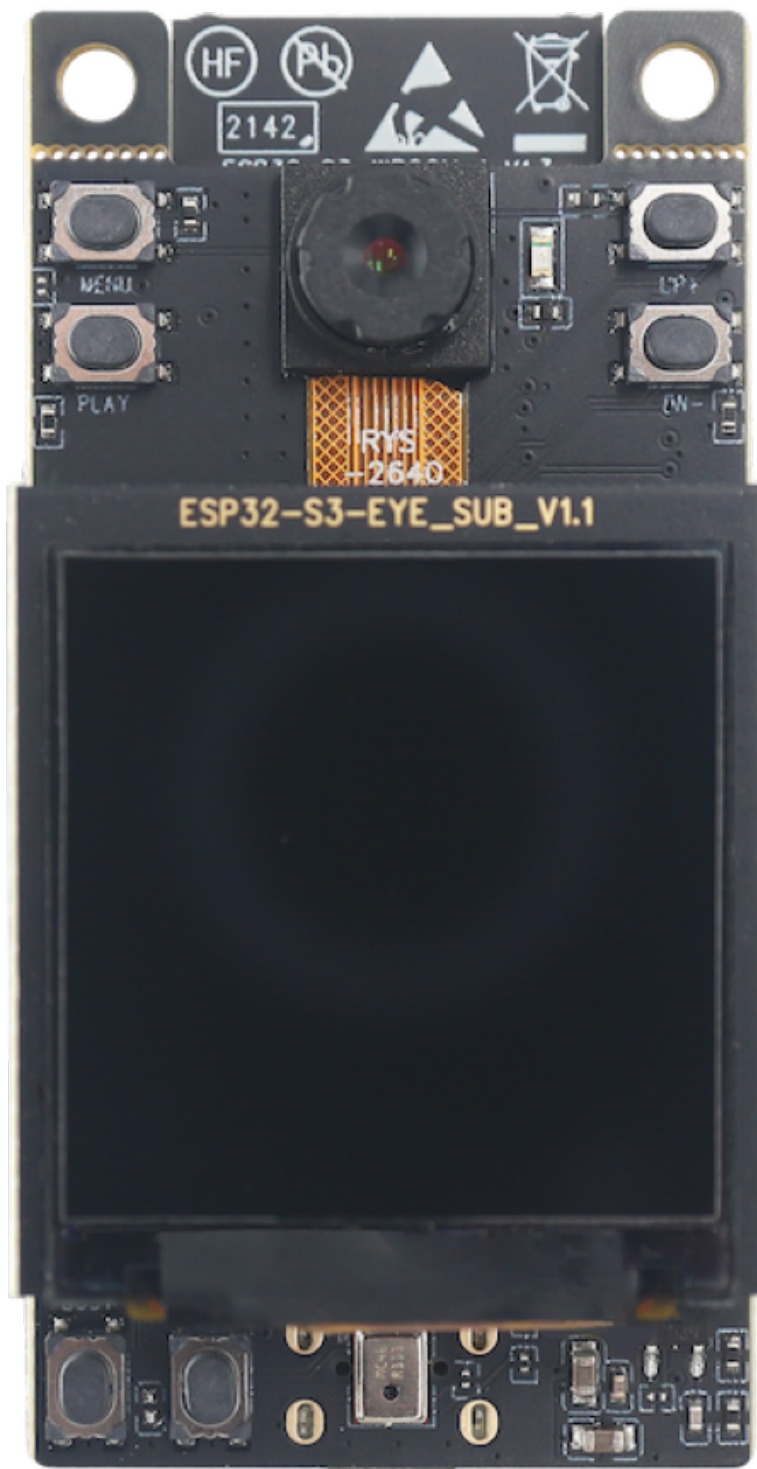
- Common peripheral interfaces such as SPI, I2S, I2C, PWM, RMT, ADC, UART, SD/MMC host controller, and TWAI controller, etc.
- Can be used for development schemes: Smart camera, face recognition, voice recognition, voice wake-up, real-time data collection and processing, complex peripheral control

Development Board:

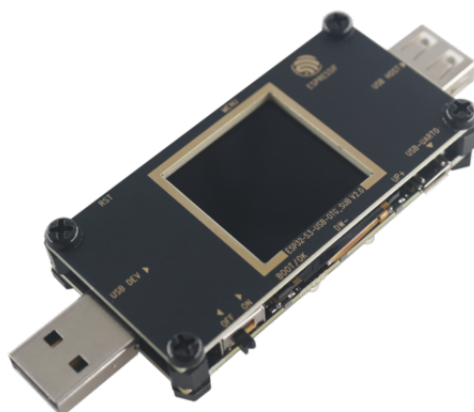
- [EESP32-S3-DevKitC-1](#) : ESP32-S3-DevKitC-1 is an entry-level development board that can be used to provision and experience examples in IDF.
- [ESP32-S3-DevKitM-1](#) : ESP32-S3-DevKitM-1 is a beginner-friendly development board that can be used to flash and experience examples in IDF.
- [ESP32-S3-BOX](#) : 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.



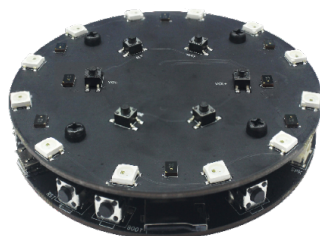
- **ESP32-S3-EYE** : ESP32-S3-EYE is a small AI (Artificial Intelligence) development board launched by Espressif. The development board is equipped with a 2-megapixel camera, an LCD display, and a microphone, suitable for applications such as image recognition and audio processing. You can use ESP-WHO 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](#)

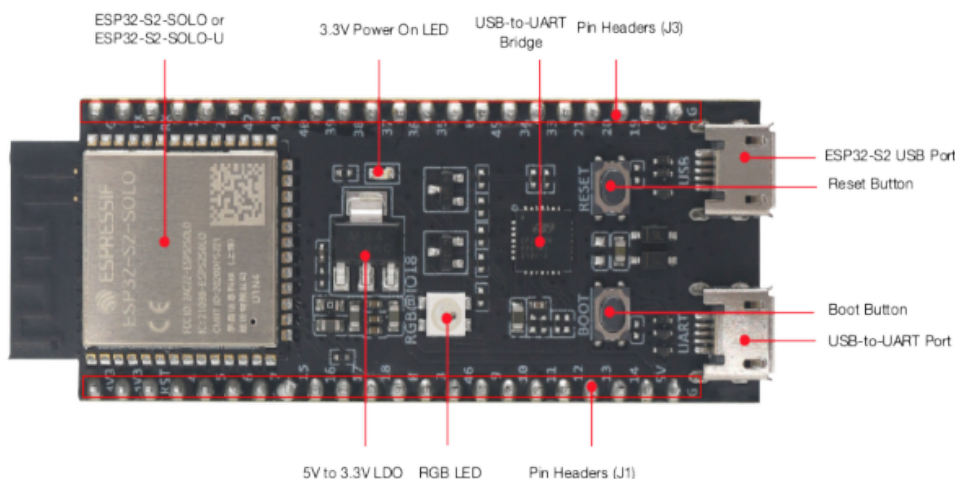
ESP32-S2

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:

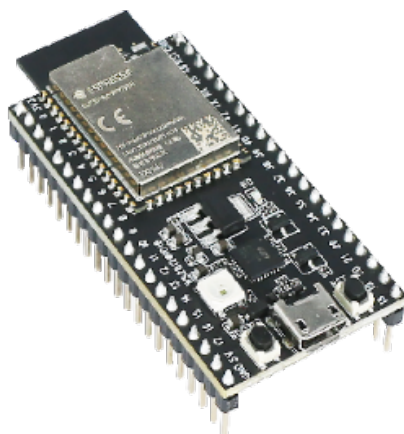
- [ESP32-S2-DevKitC-1](#) : ESP32-S2-DevKitC-1 is a beginner-level development board that can be used to flash and experience examples in IDF.



- [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](#)

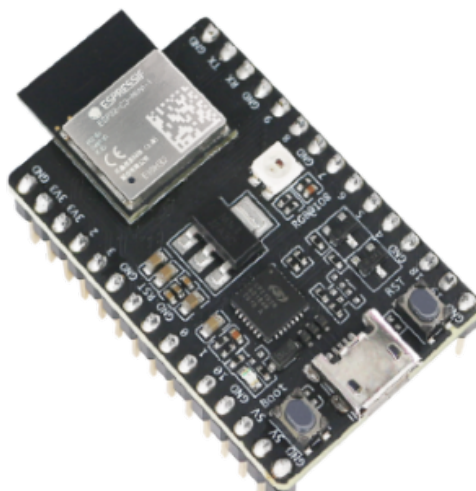
ESP32-C3

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

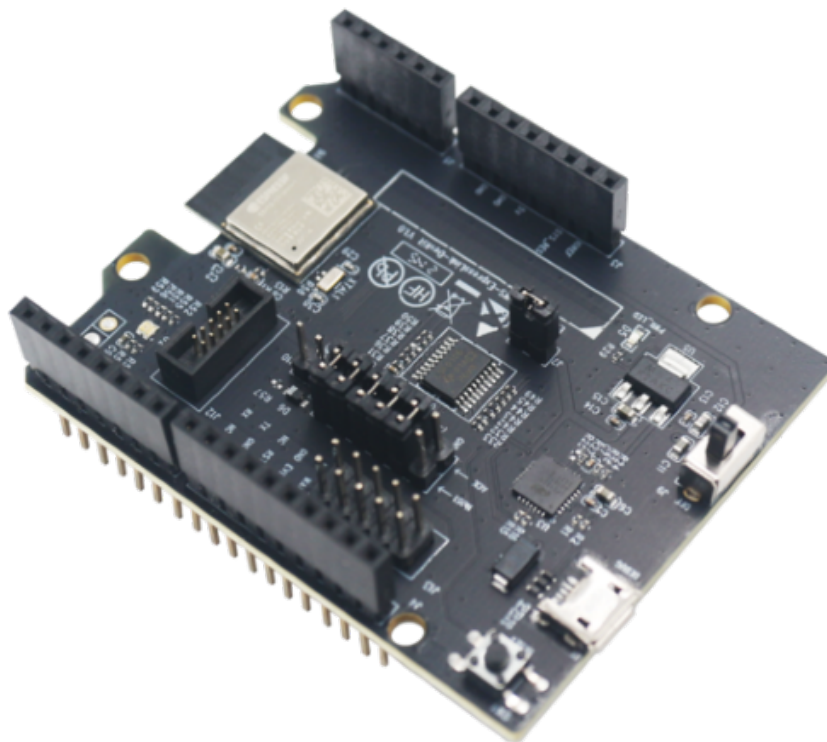
- [ESP32-C3-DevKitM-1](#) : ESP32-C3-DevKitM-1 is an entry-level development board, using the ESP32-C3-MINI-1 module, which is named for its small size. It can be used to flash and experience examples in IDF.



- [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 UART0 (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	(Enter download mode)
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 ~ 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 Official Documentation:

- [ESP8266 Datasheet](#)
- [ESP8266 Hardware Design Guidelines](#)

ESP32

The ESP32 downloads firmware via UART0 (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 Control, can't be float)
GPIO0	->	Pull Down	(Enter download mode)
GPIO2	->	Pull Down	(Default is lw level)
TXD0 (GPIO1)	->	RX	
RXD0 (GPIO3)	->	TX	

Note:

- The Strapping pin GPIO12 (MTDI) is used for selecting the VDD_SPI Flash voltage.
 - When use the 1.8V VDD_SPI Flash, the GPIO12 (MTDI) need pull up when chip power on.
 - When use the 3.3V VDD_SPI Flash, the GPIO12 (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 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 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 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 GPIO45 is used for selecting the VDD_SPI Flash voltage.
 - When use the 1.8V VDD_SPI Flash, the GPIO45 need pull up when chip power on.
 - When use the 3.3V VDD_SPI Flash, the GPIO45 need pull down when chip power on.
 - The strapping combination of GPIO46 = 1 and GPIO0 = 0 is invalid and will trigger unexpected behavior.
-

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 Official Documentation:

- [ESP32-S2 Datasheet](#)
- [ESP32-S2 Hardware Design Guidelines](#)

ESP32-C3

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

Hardware Connection When you 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)
GPIO2    -> Pull Up    (Controls the SPI Flash startup mode)
GPIO8    -> Pull Up
GPIO9    -> Pull Down (Enter download mode)
TXD0 (GPIO21) -> RX
RXD0 (GPIO20) -> TX
```

When you 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 UART0 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 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-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:

- [The hardware conditions for the ESP32-C3 chip to enter the “Download Mode”](#)
- [ESP32C3 USB & UART Download Mode](#)

官方文档:

- [ESP32-C3 Datasheet](#)
- [ESP32-C3 Hardware Design Guidelines](#)

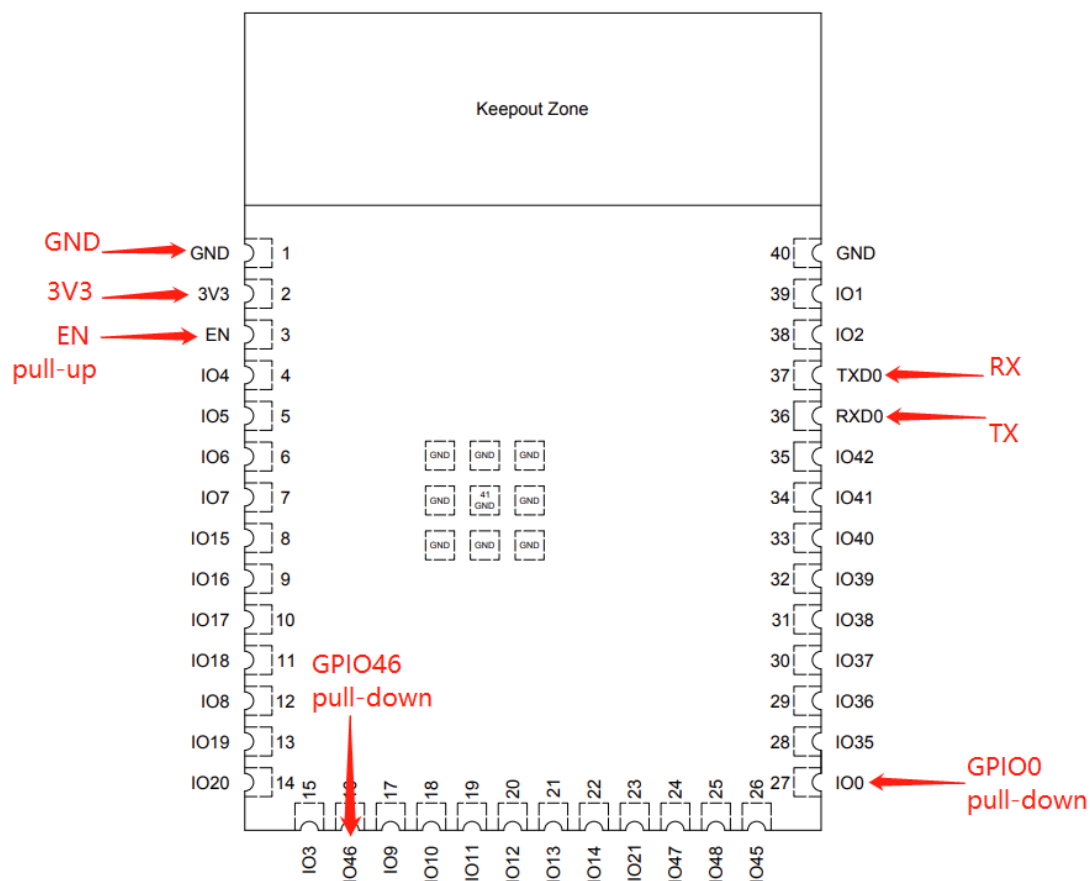
ESP32-S3

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)
GPIO0	->	Pull Down	(Enter download mode)
GPIO46	->	Pull Down	
TXD0 (GPIO43)	->	RX	
RXD0 (GPIO44)	->	TX	

For more details, please refer to the wiring diagram below:



When you 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 GPIO45 is used for selecting the VDD_SPI Flash voltage.
 - When use the 1.8V VDD_SPI Flash, the GPIO45 need pull up when chip power on.
 - When use the 3.3V VDD_SPI Flash, the GPIO45 need pull down when chip power on.
- The strapping combination of GPIO46 = 1 and GPIO0 = 0 is invalid and will trigger unexpected behavior.

Hardware conditions must be met

- ESP32-S3 chip 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.
- ESP32-S3 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 Official Documentation:

- [ESP32-S3 Datasheet](#)

- [ESP32-S3 Hardware Design Guidelines](#)

ESP32-C2

The ESP32-C2 downloads firmware via UART0 (TXD (GPIO20) and RXD (GPIO19)) 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)
GPIO8	->	Pull Up	(Default is float)
GPIO9	->	Pull Down	(Default is high level)
TXD0 (GPIO20)	->	RX	
RXD0 (GPIO19)	->	TX	

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UART0 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 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-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-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)
GPIO8	->	Pull Up	(Default is float)
GPIO9	->	Pull Down	(Default is high level)
TXD0 (GPIO24)	->	RX	
RXD0 (GPIO23)	->	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)
GPIO8	->	Pull Up	(Default is float)
GPIO9	->	Pull Down	(Default is high level)
GPIO26	->	USB_D-	
GPIO27	->	USB_D+	

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UART0 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 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.
- 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:

- [The hardware conditions for the ESP32-H2 chip to enter the “Download Mode”](#)

Official Documentation:

- [ESP32-H2 Datasheet](#)
- [ESP32-H2 Hardware Design Guidelines](#)

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)
GPIO8	->	Pull Up	(Default is float)
GPIO9	->	Pull Down	(Default is high level)
TXD0 (GPIO16)	->	RX	
RXD0 (GPIO17)	->	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)
GPIO8	->	Pull Up	(Default is float)
GPIO9	->	Pull Down	(Default is high level)
GPIO12	->	USB_D-	
GPIO13	->	USB_D+	

Note:

- After powering up the chip/module, you can verify whether it has entered Download Boot mode by using the UART0 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 $3.0\text{ V} \sim 3.6\text{ V}$; 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-C6 modules 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.

Reference Summary of Hardware Wiring Principles:

- [The hardware conditions for the ESP32-C6 chip to enter the “Download Mode”](#)

Official Documentation:

- [ESP32-C6 Datasheet](#)
- [ESP32-C6 Hardware Design Guidelines](#)

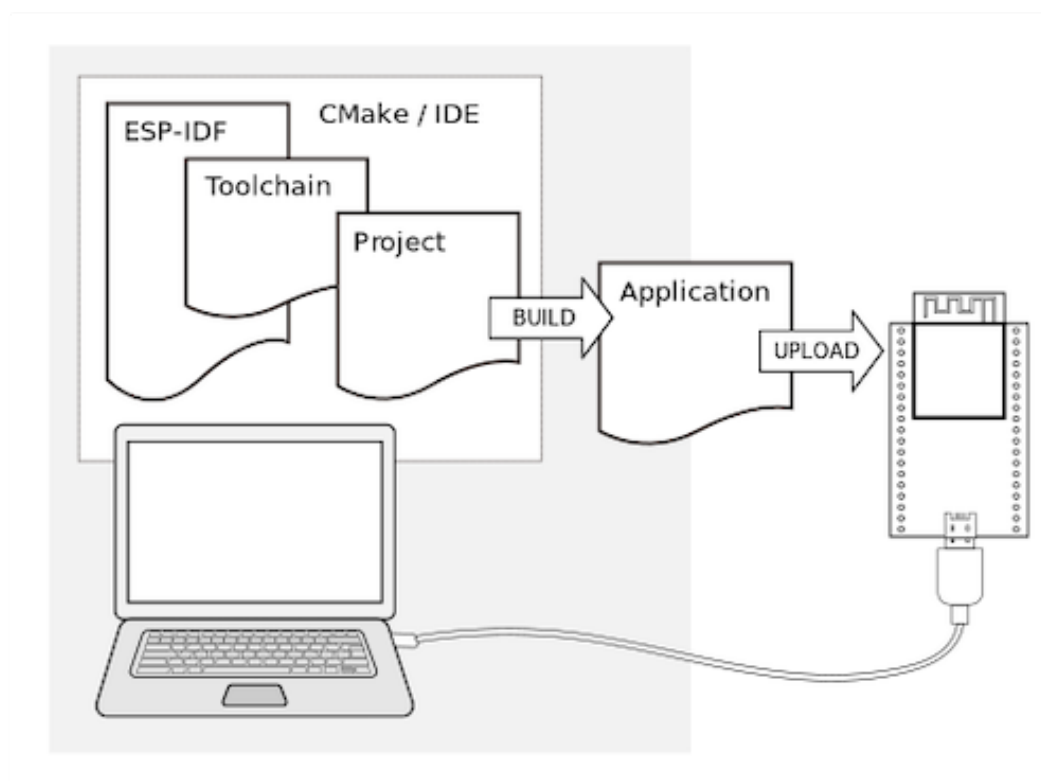
Selecting the Suitable Flashing Platform

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.

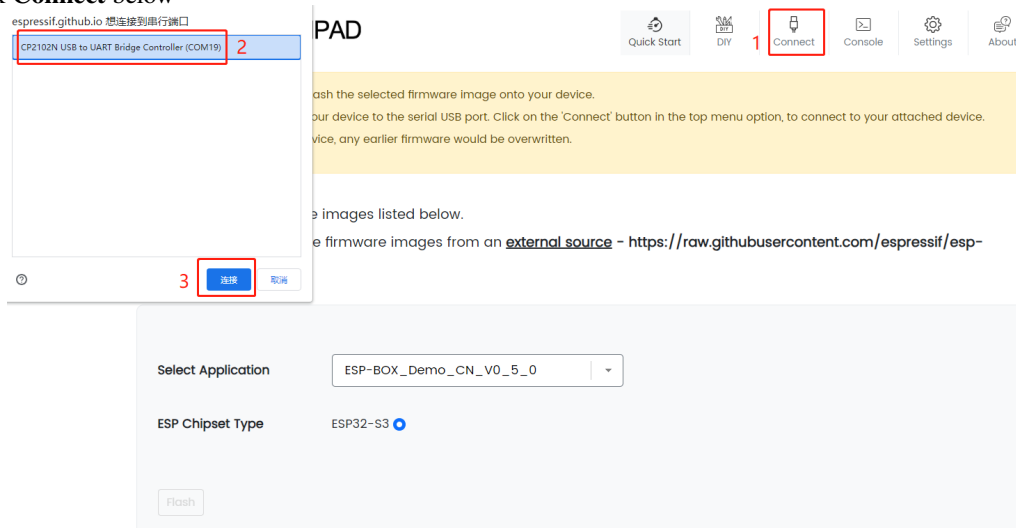
- After selecting some example firmware provided by Espressif, you can directly flash it on the web page

- Developers can also upload their compiled `.bin` files and flash them on the web page

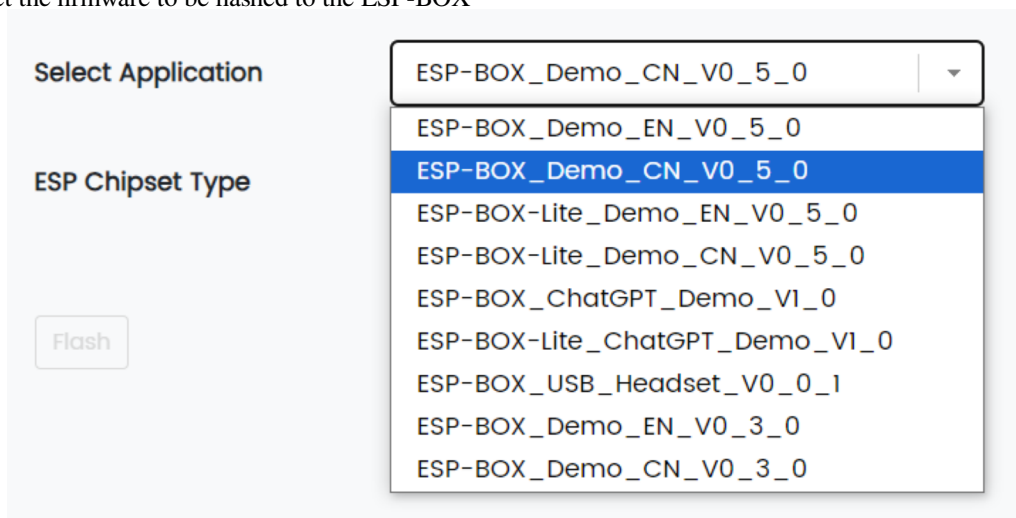
ESP LAUNCHPAD Flashing Process The following uses ESP-BOX as an example:

Open the website [ESP LAUNCHPAD](#)

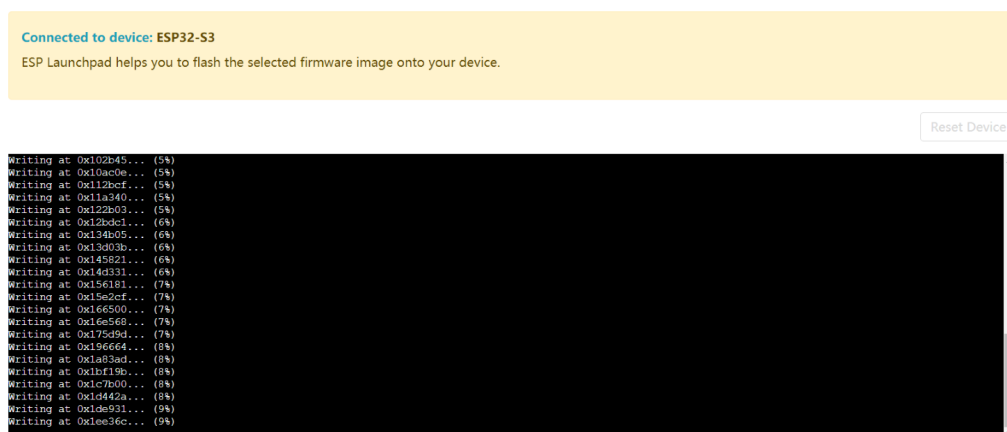
1. Click on the top right **Connect**
2. In the pop-up window, select the **Port** to connect to the ESP-BOX
3. Click **Connect** below



4. Select the firmware to be flashed to the ESP-BOX



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



- 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

Flash Download Tools Expand all + [Download selected](#)

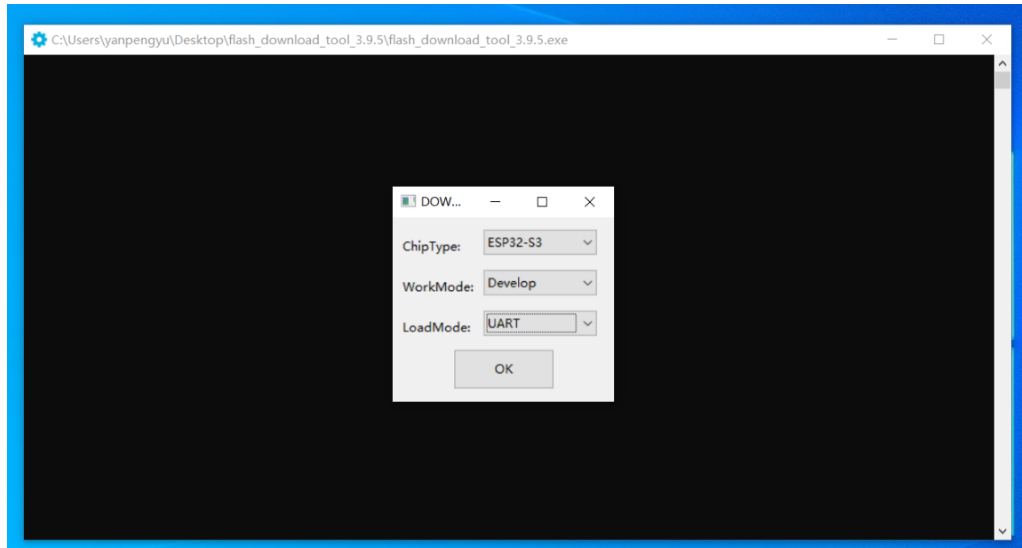
<input type="checkbox"/>	Title	Platform	Version	Release Date	Download
<input type="checkbox"/>	+ Flash Download Tools	Windows PC	V3.9.5	2023.06.12	↓

Certification and Test

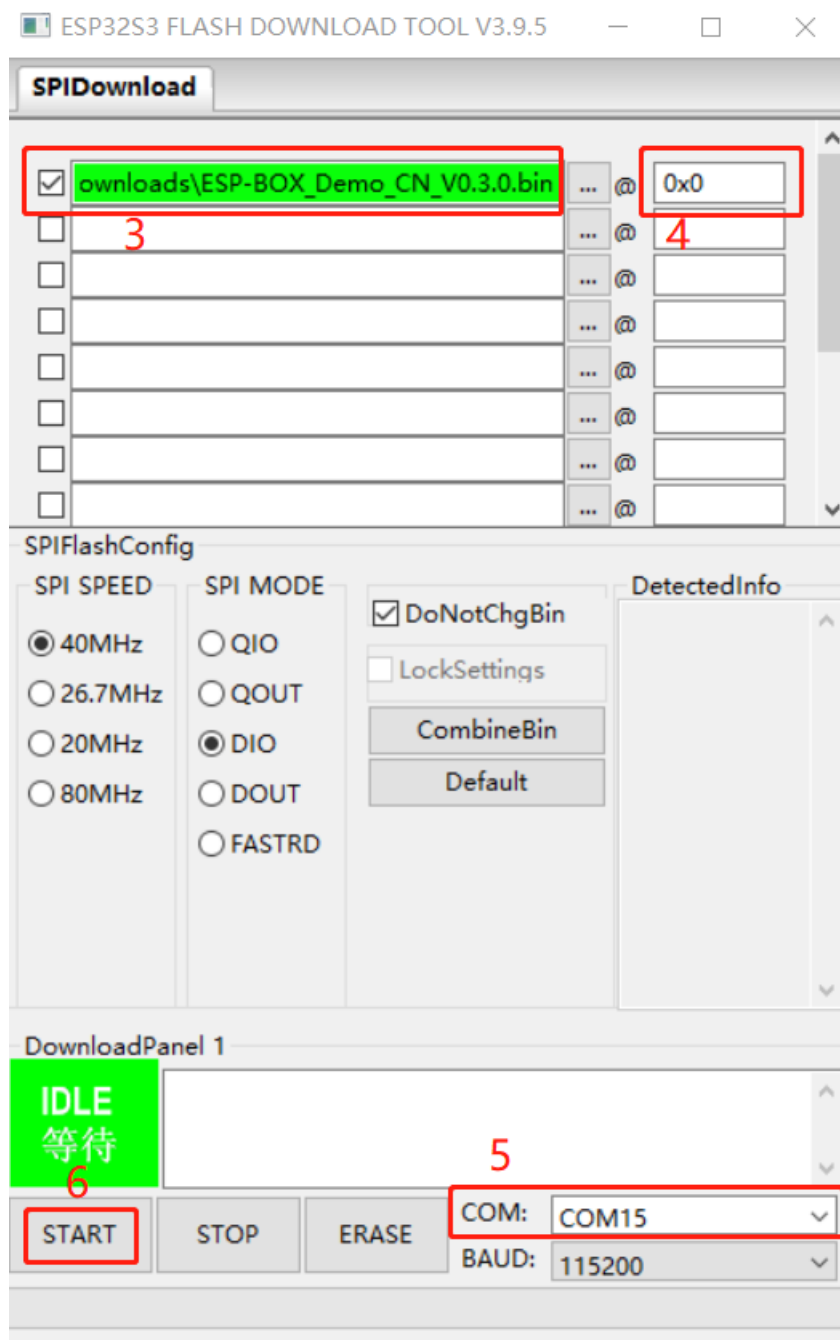
<input type="checkbox"/>	Title	Platform	Version	Release Date	Download
<input type="checkbox"/>	+ ESP RF Test Tool and Test Guide	ZIP	V2.8	2021.11.10	↓
<input type="checkbox"/>	+ ESP8266 & ESP32 WFA Certification and Test Guide	Windows PC	V1.1	2020.08.05	↓

Using the Flash download tool Flashing Process

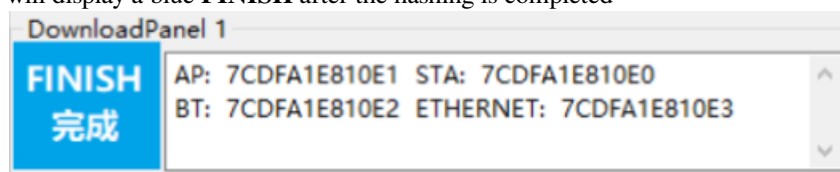
- Run the .exe file of **flash_download_tool**
- Select **Development Board (ChipType)** as the corresponding development board, there will generally be a mark on the chip. Select **Flashing Mode (LoadMode)** as UART and then click **OK** below



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 `0x0`
5. Select the **COM port** connected to the development board
6. Click **START** at the bottom left to start flashing



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
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](#)

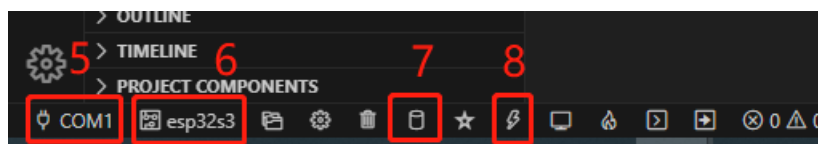
We can experience the flashing process of [ESP32-S3-BOX](#), this project uses ESP32-S3-BOX as a USB speaker.

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



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 ( 254262 remain, 26.5% used)
  .data size: 11392 bytes
  .bss size: 5408 bytes
  .text size: 73767 bytes
  .vectors size: 1027 bytes
Used Flash size : 290627 bytes
  .text : 216459 bytes
  .rodata : 73912 bytes
Total image size: 376813 bytes (.bin may be padded larger)
```

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 ✨
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:

```
C:\Espressif\tools\cmake\3.21.0\bin
C:\Espressif\tools\openocd-esp32\v0.12.0-esp32-20230419\openocd-esp32\bin
C:\Espressif\tools\ninja\1.10.2\
C:\Espressif\tools\idf-exe\1.0.3\
C:\Espressif\tools\ccache\4.8\ccache-4.8-windows-x86_64
C:\Espressif\tools\dfu-util\0.11\dfu-util-0.11-win64
C:\Espressif\frameworks\esp-idf-v5.1\tools

Checking if Python packages are up to date...
Constraint file: C:\Espressif\esp-idf\constraints.v5.1.txt
Requirement files:
- C:\Espressif\frameworks\esp-idf-v5.1\tools\requirements\requirements.core.txt
Python being checked: C:\Espressif\python_env\idf5.1_py3.11_env\Scripts\python.exe
C:\Espressif\frameworks\esp-idf-v5.1\tools\check_python_dependencies.py:12: DeprecationWarning: pkg_resources is deprecated as an API. See https://s
etuptools.pypa.io/en/latest/pkg_resources.html
  import pkg_resources
Python requirements are satisfied.

Detected installed tools that are not currently used by active ESP-IDF version.
For removing old versions of ccache, cmake, dfu-util, dist, esp-rom-elfs, esp32ulp-elf, esp32ulp-elf, idf-driver, idf-exe, idf-python-wheels, ninja,
openocd-esp32, python_env, riscv32-esp-elf, riscv32-esp-elf-gdb, tools, xtensa-clang, xtensa-esp-elf-gdb, xtensa-esp32-elf, xtensa-esp32s2-elf, x
tensa-esp32s3-elf use command 'python.exe C:\Espressif\frameworks\esp-idf-v5.1\tools\idf_tools.py uninstall'
For free up even more space, remove installation packages of those tools. Use option 'python.exe C:\Espressif\frameworks\esp-idf-v5.1\tools\idf_tool
s.py uninstall --remove-archives'.

Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

  idf.py build

C:\Espressif\frameworks\esp-idf-v5.1>
```

2. Use the `cd` command to enter a project, for example:

```
C:\Espressif\frameworks\esp-idf-v5.1>cd examples
C:\Espressif\frameworks\esp-idf-v5.1\examples>cd get-started
C:\Espressif\frameworks\esp-idf-v5.1\examples\get-started>cd hello_world
```

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:

```
MAC: a8:03:2a:ec:06:58
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x0003afff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 26640 bytes to 16668...
Writing at 0x00001000... (50 %)
Writing at 0x0000769f... (100 %)
Wrote 26640 bytes (16668 compressed) at 0x00001000 in 0.7 seconds (effective 290.2 kbit/s)...
Hash of data verified.
Compressed 174688 bytes to 97229...
Writing at 0x00010000... (16 %)
Writing at 0x0001c055... (33 %)
Writing at 0x00021a33... (50 %)
Writing at 0x0002713d... (66 %)
Writing at 0x0002d3fa... (83 %)
Writing at 0x0003517a... (100 %)
Wrote 174688 bytes (97229 compressed) at 0x00010000 in 2.5 seconds (effective 550.6 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.1 seconds (effective 454.2 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
Done
```

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)

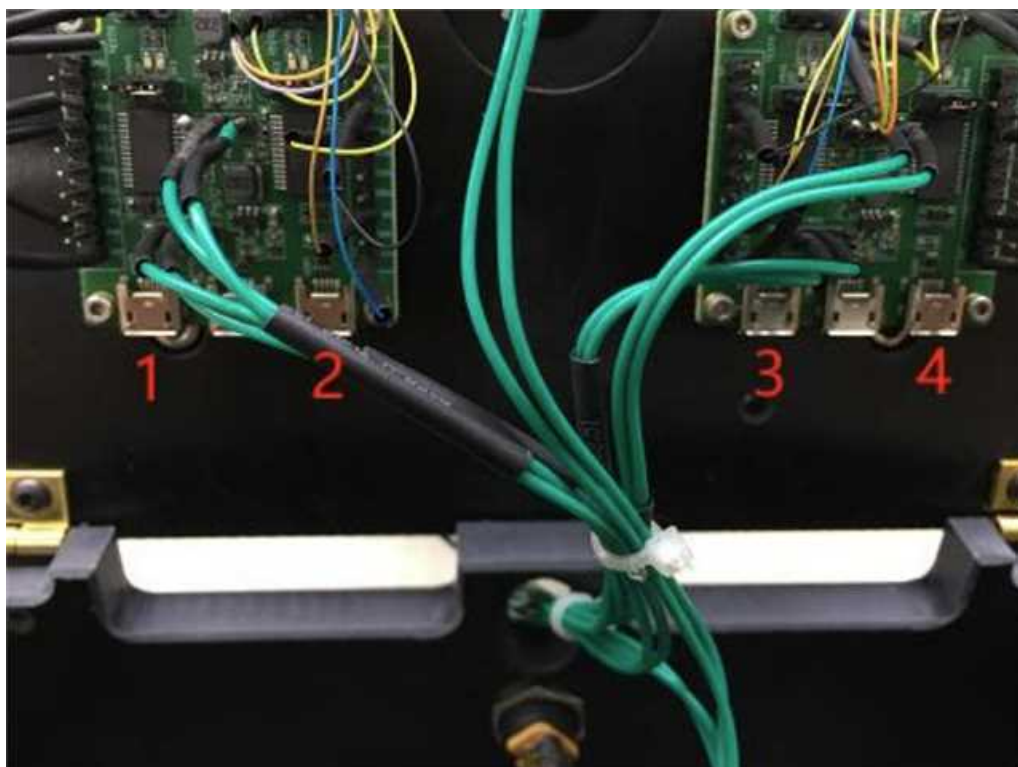
Flashing process using Espressif's official fixture

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](#)



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						
ESP32-S2						
ESP32-C3						
ESP32-S3						芯片发布公告
ESP32-C2						芯片发布公告
ESP32-C6						芯片发布公告
ESP32-H2						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 operate on multiple operating systems, including Windows, Mac, and Linux.
- Arduino IDE has extensive 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. 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.

2. 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.

3. CLion

CLion is an IDE developed by JetBrains, designed specifically for developers of C and C++ programming languages. It provides 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, making team collaboration more convenient.

Attention: MicroPython, PlatformIO, and CLion are platforms that are not directly developed by Espressif, so Espressif does not maintain issues related to these development environments. Therefore, users may encounter difficulties during use, and developers will need to seek support and solutions from relevant communities and forums.

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.
```

(continues on next page)

(continued from previous page)

```
- 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.

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:

```
void app_main(void)
{
    while (1) {
        printf("Hello world!-----\n
↪");
    }
}
```

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.

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)
{
```

(continues on next page)

(continued from previous page)

```
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/1xw <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
↳unhandled.
```

Core 0 register dump:

Stack dump detected

```
MEPC   : 0x4200696a  RA       : 0x4200696a  SP       : 0x3fc8f7c0  GP       :
↳0x3fc8b400
TP     : 0x3fc87d74  T0      : 0x4005890e  T1      : 0x3fc8f41c  T2      :
↳0x00000000
S0/FP  : 0x3c023000  S1      : 0x00000000  A0      : 0x0000000f  A1      :
↳0x3fc8f3f8
A2     : 0x00000000  A3      : 0x00000001  A4      : 0x3fc8c000  A5      :
↳0x00000000
A6     : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      :
↳0x00000000
S4     : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      :
↳0x00000000
S8     : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     :
↳0x00000000
T3     : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      :
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000005  MTVAL   :
↳0x00000001
```

- XTENSA

```
Guru Meditation Error: Core  0 panic'ed (LoadProhibited). Exception 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     : 0x0000000c  A8     : 0x00000000  A9     :
↳0x3fc97d20
A10    : 0x0000000f  A11    : 0x3fc97fac  A12    : 0x3fc97d70  A13    :
↳0x0000000c
A14    : 0x3fc941e4  A15    : 0x00000000  SAR    : 0x00000004  EXCCAUSE:
↳0x0000001c
```

```
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
l8ui 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(0x00000000) 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.

The screenshot displays a debugger interface with two main panels. The left panel shows assembly code with addresses and instructions, such as:

```
4038fa60: 080c      addi   a1,sp,16
4038fa68: 3fca47b7  lut   a5,0x3fca4
4038faec: 3587a503  lw    a0,856(a5) # 3fca4358 <xTimerQueue>
4038faf0: adcfd0ef  jal   ra,4038cdc0 <xQueueReceive>
4038faf4: c571     beqz  a0,4038fbc0 <prvProcessReceivedComman
ds+0xf2>
4038faf6: 47c2     lw    a5,16(sp)
4038faf8: fe07d3e3  bgez  a5,4038fade <prvProcessReceivedComman
ds+0x1e>
4038fafc: bf69     j     4038fad6 <prvProcessReceivedComman
ds+0x8>
4038fafe: 4462     lw    s0,24(sp)
4038fb00: 485c     lw    a5,20(s0)
4038fb02: c789     beqz  a5,4038fb0c <prvProcessReceivedComman
ds+0x3e>
4038fb04: 00440513  addi  a0,s0,4
4038fb08: bf5fc0ef  jal   ra,4038c6fc <uxListRemove>
4038fb0c: 0000     addi  a0,sp,12
4038fb0e: ebbff0ef  jal   ra,4038f9c8 <prvSampleTimeNow>
4038fb12: 47c2     lw    a5,16(sp)
4038fb14: 4715     li    a4,5
4038fb16: 08e78863  beq   a5,a4,4038fba6 <prvProcessReceivedCom
mands+0xd8>
4038fb18: 04f74963  blt   a4,a5,4038fb6c <prvProcessReceivedCom
mands+0x9e>
4038fb1e: 470d     li    a4,3
4038fb20: 06e78d63  beq   a5,a4,4038fb9a <prvProcessReceivedCom
mands+0xcc>
```

The right panel shows a crash log with the following key information:

```
51 Guru Meditation Error: Core 0 panic'ed (Load access fault). Exception was unhandled.
52
53 Core 0 register dump:
54 MEPC : 0x4038fb00 RA : 0x4038faf4 SP : 0x3fcae800 GP : 0x3fc99a00
55 0x4038fb00: prvProcessReceivedCommands at D:/esp_king/esp-ldf20230608/esp-ldf/components/freer
56
57 0x4038faf4: prvProcessReceivedCommands at D:/esp_king/esp-ldf20230608/esp-ldf/components/freer
58
59 TP : 0x3fc08454 T0 : 0x00000000 T1 : 0x00000000 T2 : 0x00000000
60 S0/FP : 0x00000000 S1 : 0x00000000 A0 : 0x00000001 A1 : 0x3fc9eaf8
61 A2 : 0x4038c57a A3 : 0x3fcae880 A4 : 0x3fca4000 A5 : 0x00000004
62 0x4038c57a: xEventGroupClearBits at D:/esp_king/esp-ldf20230608/esp-ldf/components/freertos/ev
63
64 A6 : 0x00000000 A7 : 0x00000000 S2 : 0x00000000 S3 : 0x00000000
65 S4 : 0x00000000 S5 : 0x00000000 S6 : 0x00000000 S7 : 0x00000000
66 S8 : 0x00000000 S9 : 0x00000000 S10 : 0x00000000 S11 : 0x00000000
67 T3 : 0x00000000 T4 : 0x00000000 T5 : 0x00000000 T6 : 0x00000000
68 MSTATUS : 0x00001881 MTVEC : 0x40380001 MCAUSE : 0x00000005 MTVAL : 0x00000004
69 0x40380001: x_vector_table at ???
70
71 MHARTID : 0x00000000
72 Backtrace: 0x4038fb00:0x3fcae860 0x4038fbda:0x3fcae890 0x4038fcd8:0x3fcae8b0
73 0x4038fb00: prvProcessReceivedCommands at D:/esp_king/esp-ldf20230608/esp-ldf/components/freer
74
75 0x4038fbda: prvTimerTask at D:/esp_king/esp-ldf20230608/esp-ldf/components/freertos/timers.c:
476
```

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.

```

}
// If there's no callback, we just need a delay.
if(pTimerCallback == NULL) {
    vTaskDelay(period);
} else {
    // If it doesn't exist, create it.
    if(g_timer != NULL) {
        g_timer = xTimerCreate("NFCTimer", 10, pdFALSE, pTimerCallback, timer_expiry_cb);
    }

    // Set the period, then start/restart it.
    xTimerChangePeriod(g_timer, period, BLOCK_TIME);
    xTimerReset(g_timer, BLOCK_TIME);
}
}

}
// If there's no callback, we just need a delay.
if(pTimerCallback == NULL) {
    vTaskDelay(period);
} else {
    // If it doesn't exist, create it.
    if(g_timer == NULL) {
        g_timer = xTimerCreate("NFCTimer", 10, pdFALSE, pTimerCallback, timer_expiry_cb);
    }

    // Set the period, then start/restart it.
    xTimerChangePeriod(g_timer, period, BLOCK_TIME);
    xTimerReset(g_timer, BLOCK_TIME);
}
}

```

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:
PC      : 0x400358e6  PS      : 0x00060533  A0      : 0x80376ae6  A1      : 0x3fca2590
0x400358e6: rom_i2c_writeReg_Mask in ROM

A2      : 0x0000006d  A3      : 0x00000001  A4      : 0x00000004  A5      : 0x00000004
A6      : 0x00000000  A7      : 0x0000001d  A8      : 0x3fcef3d4  A9      : 0x00000000
A10     : 0x00060523  A11     : 0x00000004  A12     : 0x00060524  A13     : 0x00000001
A14     : 0x00000001  A15     : 0x3fc98824  SAR     : 0x0000000a  EXCCAUSE: 0x0000001c
EXCVADDR: 0x00000190  LBEG    : 0x40056f5c  LEND    : 0x40056f72  LCOUNT : 0x00000000
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 `0x0000000a` 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:
Stack dump detected
MEPC   : 0x42006960  RA       : 0x42006984  SP       : 0x3fc8f7c0  GP       : _
↳0x3fc8b400
TP     : 0x3fc87d7c  T0      : 0x4005890e  T1      : 0x20000000  T2      : _
↳0x00000000
S0/FP  : 0x3c023000  S1      : 0x00000000  A0      : 0x00000000  A1      : _
↳0x3fc8f3f8
A2     : 0x00000000  A3      : 0x00000001  A4      : 0x3fc8c000  A5      : _
↳0x0000005f
A6     : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      : _
↳0x00000000
S4     : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      : _
↳0x00000000
S8     : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : _
↳0x00000000
T3     : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      : _
↳0x00000000
MSTATUS: 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   : _
↳0x0000000a
```

• 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 `0x0000000a` stored in `EXCVADDR` indicates that it wants to access the content in an array or structure starting from `NULL` address.

```
Guru Meditation Error: Core  0 panic'ed (StoreProhibited). Exception was unhandled.

Core  0 register dump:
PC     : 0x42007c21  PS     : 0x00060630  A0      : 0x82007c38  A1      : _
↳0x3fc97d70
A2     : 0x00000000  A3     : 0x3c023f48  A4      : 0x3c023fc8  A5      : _
↳0x3fc97d90
A6     : 0x3fc97d70  A7     : 0x0000000c  A8      : 0x0000005f  A9      : _
↳0x3fc97d00
A10    : 0x00000031  A11    : 0x3fc97fac  A12     : 0x3fc97d70  A13     : _
↳0x0000000c
A14    : 0x3fc941e4  A15    : 0x00000000  SAR     : 0x00000004  EXCCAUSE:_
↳0x0000001d
EXCVADDR: 0x0000000a  LBEG   : 0x400556d5  LEND    : 0x400556e5  LCOUNT  : _
↳0xffffffff

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 (*fp_ptr_t)(void);
volatile fp_ptr_t fp_ptr = (fp_ptr_t) 0x4;
fp_ptr();
```

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

```
42006958:      1101          addi    sp, sp, -32
4200695a:      ce06          sw     ra, 28(sp)
4200695c:      4791          li     a5, 4
4200695e:      c63e          sw     a5, 12(sp)
42006960:      47b2          lw     a5, 12(sp)
42006962:      9782          jalr   a5
42006964:      40f2          lw     ra, 28(sp)
42006966:      6105          addi   sp, sp, 32
42006968:      8082          ret
```

Exception Phenomenon

- RISC-V

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

```
Core  0 register dump:
```

```
Stack dump detected
```

```
MEPC   : 0x00000004  RA       : 0x42006964  SP       : 0x3fc8f7b0  GP       :
↳0x3fc8b400
TP     : 0x3fc87d8c  T0      : 0x4005890e  T1      : 0x20000000  T2      :
↳0x00000000
S0/FP  : 0x3c023000  S1      : 0x00000000  A0      : 0x00000031  A1      :
↳0x3fc8f3f8
```

(continues on next page)

(continued from previous page)

```

A2      : 0x00000000  A3      : 0x00000001  A4      : 0x3fc8c000  A5      : _
↳0x00000004
A6      : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      : _
↳0x00000000
S4      : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      : _
↳0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : _
↳0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      : _
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000001  MTVAL   : _
↳0x00000004

```

- Xtensa

```

Guru Meditation Error: Core  0 panic'ed (InstrFetchProhibited). Exception was_
↳unhandled.

Core  0 register dump:
PC      : 0x00000004  PS      : 0x00060630  A0      : 0x800d539a  A1      : _
↳0x3ffb4a30
A2      : 0x400de738  A3      : 0x3f40379c  A4      : 0x3f40381c  A5      : _
↳0x3ffb4a60
A6      : 0x3ffb4a40  A7      : 0x0000000c  A8      : 0x800e4dbe  A9      : _
↳0x3ffb49d0
A10     : 0x00000031  A11     : 0x3ffaafc64  A12     : 0x3ffb4a40  A13     : _
↳0x0000000c
A14     : 0x3ffb2770  A15     : 0x0000cdcd  SAR     : 0x00000004  EXCCAUSE:_
↳0x00000014
EXCVADDR: 0x00000004  LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT : _
↳0xffffffff

Backtrace: 0x00000001:0x3ffb4a30 0x400d5397:0x3ffb4a60 0x400e54f8:0x3ffb4a80_
↳0x400858e9:0x3ffb4ab0

```

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  0 register dump:
PC      : 0x08a408a4  PS      : 0x00050033  A0      : 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: 0x08a408a1:0x3fcada60 0x08a408a1:0x08a408a4 |<-CORRUPTED

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:

```

uint32_t instr_dram_addr = 0x0;
intptr_t instr_addr = MAP_DRAM_TO_IRAM((intptr_t)&instr_dram_addr);
typedef void (*illegal_instr_t)(void);
illegal_instr_t illegal_instr = (uint32_t *)instr_addr; // Ensure that the memory_
↳data in instr_addr is 0x0
illegal_instr();

```

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 Meditation Error: Core  0 panic'ed (Illegal instruction). Exception was_
↳unhandled.

Core  0 register dump:
Stack dump detected
MEPC    : 0x4005f002  RA      : 0x42006962  SP      : 0x3fc8f7c0  GP      :_
↳0x3fc8b400
0x42006962: illegal_instr_fault at /home/jacques/useful_example/crash_sim/store_
↳access_fault/main/hello_world_main.c:79

TP      : 0x3fc87d8c  T0      : 0x4005890e  T1      : 0x20000000  T2      :_
↳0x00000000
S0/FP   : 0x3c023000  S1      : 0x00000000  A0      : 0x3fc8f840  A1      :_
↳0x3fc8f3f8
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x3fc8c000  A5      :_
↳0x4005f000
A6      : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      :_
↳0x00000000

```

(continues on next page)

(continued from previous page)

```

S4      : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      : _
↳0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : _
↳0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      : _
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000002  MTVAL   : _
↳0x00000000

```

- Xtensa

```

Guru Meditation Error: Core  0 panic'ed (IllegalInstruction). Exception was _
↳unhandled.
Memory dump at 0x403e298c: 00000000 00000000 00000000
Core  0 register dump:
PC      : 0x403e2990  PS      : 0x00060130  A0      : 0x820059fe  A1      : _
↳0x3fcf2990
A2      : 0x4200c030  A3      : 0x3c0232fc  A4      : 0x3c02337c  A5      : _
↳0x3fcf29c0
A6      : 0x3fcf29a0  A7      : 0x0000000c  A8      : 0x820059f4  A9      : _
↳0x3fcf2930
A10     : 0x00000031  A11     : 0x3fcf2be4  A12     : 0x3fcf29a0  A13     : _
↳0x0000000c
A14     : 0x3fc922c4  A15     : 0x00000000  SAR     : 0x00000004  EXCCAUSE:_
↳0x00000000
EXCVADDR: 0x00000000  LBEG    : 0x400556d5  LEND    : 0x400556e5  LCOUNT : _
↳0xffffffff
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.

Scenarios 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 scenario 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 ([CONFIG_ESP_SLEEP_POWER_DOWN_FLASH](#)) and enabling the CONFIG CS software pull-up during sleep ([CONFIG_ESP_SLEEP_FLASH_LEAKAGE_WORKAROUND](#)), the problem no longer occurs.


```

test sleep battery_read_key before
battery_read_key:3
is_wifi_enable=0 is_standalone=0
test sleep UPM_init before
test 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 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

S0/FP   : 0x3c0e6000 S1     : 0x3c0e6000 A0     : 0x3c0e5d8c A1     : 0x3fca4958
A2      : 0x00000000 A3     : 0x00000001 A4     : 0x600c0000 A5     : 0x00000000
A6      : 0x00000010 A7     : 0x000000c8 S2     : 0x00000000 S3     : 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 (including `mmap`), 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 RA      : 0x40381e82 SP      : 0x3fc8f7c0 GP      : _
↳0x3fc8b400
0x40381e82: cache_access_test_func at /home/jacques/useful_example/crash_sim/store_
↳access_fault/main/hello_world_main.c:38

TP      : 0x3fc87d5c T0     : 0x4005890e T1     : 0x20000000 T2     : _
↳0x00000000
S0/FP   : 0x3c023000 S1     : 0x00000000 A0     : 0x3c025610 A1     : _
↳0x00000000
A2      : 0x00000000 A3     : 0x00000200 A4     : 0x600c4000 A5     : _
↳0x3c02561c
A6      : 0x60023000 A7     : 0x0000000a S2     : 0x00000000 S3     : _
↳0x00000000
S4      : 0x00000000 S5     : 0x00000000 S6     : 0x00000000 S7     : _
↳0x00000000

```

(continues on next page)

(continued from previous page)

```

S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : 0x00000000
↳0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      : 0x00000000
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000019  MTVAL   : 0x6944806f
↳0x6944806f

```

- Xtensa

```

Guru Meditation Error: Core  0 panic'ed (Cache disabled but cached memory region
↳accessed) .

Core  0 register dump:
PC      : 0x400830ed  PS      : 0x00060a34  A0      : 0x800d539a  A1      : 0x3ffb4a40
↳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      : 0x3ffb4a60
↳0x3ffb4a60
0x400de738: vprintf at /builds/idf/crosstool-NG/.build/xtensa-esp32-elf/src/newlib/
↳newlib/libc/stdio/vprintf.c:30

A6      : 0x3ffb4a40  A7      : 0x0000000c  A8      : 0x3f4040c4  A9      : 0x3ffb4a10
↳0x3ffb4a10
A10     : 0x00000001  A11     : 0xbad00bad  A12     : 0x3ffb2608  A13     : 0x0000000c
↳0x0000000c
A14     : 0x3ffb2770  A15     : 0x0000cdcd  SAR     : 0x00000004  EXCCAUSE: 0x00000007
↳0x00000007
EXCVADDR: 0x00000000  LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT : 0xfffffff
↳0xfffffff

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:

```

IDiff Time: 180
(00:00:03.131) NVM_TASK: InitDiff Time: 179
Initializing NVS Flash
Diff Time: 179
Diff Time: 183
Diff Time: 183
Diff Time: 183
Guru Meditation Error: Core  0 panic'ed (Cache disabled but cached memory region accessed).

Core  0 register dump:
PC      : 0x40086728  PS      : 0x00060035  A0      : 0x800868a0  A1      : 0x3ffbfb40
0x40086728: gpio_set_level at /home/jacques/sdk/esp-idf/components/driver/gpio/gpio.c:237

A2      : 0x3ffbeff8  A3      : 0x3ffc1126  A4      : 0x3ffc111c  A5      : 0x3ffc1126
A6      : 0x00000000  A7      : 0x0006a0d9  A8      : 0x800842d8  A9      : 0x00000002
A10     : 0x000000d0  A11     : 0x00000001  A12     : 0x00000000  A13     : 0x80000000
A14     : 0x3ffc0a20  A15     : 0x00000000  SAR     : 0x00000005  EXCCAUSE: 0x00000007
EXCVADDR: 0x00000000  LBEG    : 0x40082e88  LEND    : 0x40082e90  LCOUNT : 0x00000026
0x40082e88: esp_timer_impl_get_counter_reg at /home/jacques/sdk/esp-idf/components/esp_timer/src/esp_timer_impl_lac.c:118
0x40082e90: esp_timer_impl_get_counter_reg at /home/jacques/sdk/esp-idf/components/esp_timer/src/esp_timer_impl_lac.c:128

Core  0 was running in ISR context:
EPC1    : 0x400d3633  EPC2    : 0x00000000  EPC3    : 0x4008b20c  EPC4    : 0x00000000
0x400d3633: uart_hal_write_txfifo at /home/jacques/sdk/esp-idf/components/hal/uart_hal_iram.c:26
0x4008b20c: spi_flash_ll_cmd_is_done at /home/jacques/sdk/esp-idf/components/hal/esp32/include/hal/spi_flash_ll.h:77
(inlined by) spi_flash_hal_poll_cmd_done at /home/jacques/sdk/esp-idf/components/hal/spi_flash_hal_common.inc:38

Backtrace: 0x40086725:0x3ffbfb40 0x4008689d:0x3ffbfb70 0x400846bd:0x3ffbfb0 0x4008b206:0x3ffd6700 0x4008b78e:0x3ffd6730 0x40086850:0x3ffd6880 0x400ef6f5:0x3ffd68d0 0x400efaa3:0x3ffd68d0 0x400ed6d5:0x3ffd6940 0x400ee08b:0x3ffd69a0 0x400ee14f:0x3ffd69e0 0x400ec32f:0x40086725: gpio_intr_disable at /home/jacques/sdk/esp-idf/components/driver/gpio/gpio.c:191
0x4008689d: gptimer_default_isr at /home/jacques/sdk/esp-idf/components/driver/gptimer/gptimer.c:543
0x400846bd: _xt_lowint1 at /home/jacques/sdk/esp-idf/components/xtensa/xtensa_vectors.S:1240
0x4008b206: spi_flash_hal_poll_cmd_done at /home/jacques/sdk/esp-idf/components/hal/spi_flash_hal_common.inc:37
0x4008b78e: spi_flash_hal_read at /home/jacques/sdk/esp-idf/components/hal/spi_flash_hal_common.inc:194
0x4008df15: spi_flash_chip_generic_read at /home/jacques/sdk/esp-idf/components/spi_flash/spi_flash_chip_generic.c:246
0x40085a22: esp_flash_read at /home/jacques/sdk/esp-idf/components/spi_flash/esp_flash_api.c:899
0x400e96ff: esp_partition_read_raw at /home/jacques/sdk/esp-idf/components/esp_partition/partition_target.c:102
0x400edc8f: nvs::NVSPartition::read_raw(unsigned int, void*, unsigned int) at /home/jacques/sdk/esp-idf/components/nvs_flash/nvs_flash.cpp:102
0x400ef6f5: nvs::Page::load(nvs::Partition*, unsigned long, unsigned long) at /home/jacques/sdk/esp-idf/components/nvs_flash/src/nvs_page.cpp:102
0x400efaa3: nvs::PageManager::load(nvs::Partition*, unsigned long, unsigned long) at /home/jacques/sdk/esp-idf/components/nvs_flash/src/nvs_page_manager.cpp:102

```

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.

```

5 //static bool LEDsTimer_on_alarm(gptimer_handle_t LEDtimer_hdl, const gptimer_alarm_ev
5 static bool IRAM_ATTR LEDsTimer_on_alarm(gptimer_handle_t LEDtimer_hdl, const gptimer
7 {
3     int64_t start, diff;
3     start = esp_timer_get_time();
3     BaseType_t high_task_awoken = pdFALSE;
1     /**LLL TEST 12.04.2024
2     QueueHandle_t queue = (QueueHandle_t)user_data;
3     *****/
4     uint8_t count;
5     static bool pin_level = 0;
5
7     gpio_set_level( IO0_PIN, 1 );
3     ledc_timer_rst( ledc_timer.speed_mode, ledc_timer.timer_num ); //To synchronized
3
3 #if 1
1     /*--- Switch OFF all COMMON */
2     for ( count = 0; count < GPIO_NUM_MAX_ROW; count++ )
3     {
4         gpio_set_level( LED_matrix.row_GPIO[count], 0 );
5     }
5
7     /*--- Set the LED's duty cycle */
3     for ( count = 0; count < GPIO_NUM_MAX_COL; count++ )
3     {
3         //uint8_t index = row_count * GPIO_NUM_MAX_COL + count;
1         /** Not SAFE */
2         ledc_set_duty(ledc_channel[count].speed_mode, ledc_channel[count].channel, dut
3         ledc_update_duty(ledc_channel[count].speed_mode, ledc_channel[count].channel);
4         /*******/
5         /** SAFE but doesn't work properly and not putted in IRAM *
5         ledc_set_duty_and_update(ledc_channel[count].speed_mode, 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
↳unhandled.

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
TP     : 0x3fc87d4c  T0      : 0x4005890e  T1      : 0x20000000  T2      :
↳0x00000000
S0/FP  : 0x3c023000  S1      : 0x00000000  A0      : 0x0000000c  A1      :
↳0x00000000
A2     : 0x00000200  A3      : 0x00000200  A4      : 0x600c4000  A5      :
↳0x00000000

```

(continues on next page)

(continued from previous page)

A6	: 0x60023000	A7	: 0x0000000a	S2	: 0x00000000	S3	: _
	↪0x00000000						
S4	: 0x00000000	S5	: 0x00000000	S6	: 0x00000000	S7	: _
	↪0x00000000						
S8	: 0x00000000	S9	: 0x00000000	S10	: 0x00000000	S11	: _
	↪0x00000000						
T3	: 0x00000000	T4	: 0x00000000	T5	: 0x00000000	T6	: _
	↪0x00000000						
MSTATUS	: 0x00001881	MTVEC	: 0x40380001	MCAUSE	: 0x00000002	MTVAL	: _
	↪0x00000000						

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:

```
static portMUX_TYPE s_init_spinlock = portMUX_INITIALIZER_UNLOCKED;
portENTER_CRITICAL(&s_init_spinlock);
while (1) {
}
```

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  T0     : 0x00000000  T1     : 0x00000000  T2     : _
↪0x00000000
S0/FP  : 0x00000000  S1     : 0x00000000  A0     : 0x00000001  A1     : _
↪0x00000000
A2     : 0x00000000  A3     : 0x00000004  A4     : 0x00000001  A5     : _
↪0x3fc8c000
A6     : 0x00000000  A7     : 0x00000000  S2     : 0x00000000  S3     : _
↪0x00000000
S4     : 0x00000000  S5     : 0x00000000  S6     : 0x00000000  S7     : _
↪0x00000000
S8     : 0x00000000  S9     : 0x00000000  S10    : 0x00000000  S11    : _
↪0x00000000
T3     : 0x00000000  T4     : 0x00000000  T5     : 0x00000000  T6     : _
↪0x00000000
MSTATUS : 0x00001881  MTVEC  : 0x40380001  MCAUSE : 0x00000018  MTVAL  : _
↪0x0000a001
```

- Xtensa

```

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

Core  0 register dump:
PC      : 0x40378b06  PS      : 0x00060634  A0      : 0x82002239  A1      : 0x3fc984f0
↳0x3fc984f0
0x40378b06: esp_cpu_wait_for_intr at /home/jacques/sdk/esp-idf/components/esp_hw_
↳support/cpu.c:110

A2      : 0x00000000  A3      : 0x00000000  A4      : 0x3fc96c70  A5      : 0x3fc96c50
↳0x3fc96c50
A6      : 0x42005e10  A7      : 0x00000000  A8      : 0x82009452  A9      : 0x3fc984b0
↳0x3fc984b0
0x42005e10: timer_task at /home/jacques/sdk/esp-idf/components/esp_timer/src/esp_
↳timer.c:470

A10     : 0x00000000  A11     : 0x00000000  A12     : 0x3fc96c50  A13     : 0x3fc96c30
↳0x3fc96c30
A14     : 0x00000000  A15     : 0x00000000  SAR     : 0x00000000  EXCCAUSE: 0x00000005
↳0x00000005
EXCVADDR: 0x00000000  LBEG    : 0x00000000  LEND    : 0x00000000  LCOUNT : 0x00000000
↳0x00000000

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:
PC      : 0x42007c32  PS      : 0x00060034  A0      : 0x80379e74  A1      : 0x3fc99f20
↳0x3fc99f20
0x42007c32: interrupt_wdt_fault_task at /home/jacques/useful_example/crash_test/
↳main/hello_world_main.c:121 (discriminator 1)

A2      : 0x00000000  A3      : 0x00000000  A4      : 0x00000001  A5      : 0x00000000
↳0x00000000
A6      : 0x00000001  A7      : 0x00000000  A8      : 0x82007c32  A9      : 0x3fc99ef0
↳0x3fc99ef0
A10     : 0x00000001  A11     : 0x3fc93a54  A12     : 0x00060020  A13     : 0x00060023
↳0x00060023
A14     : 0x00060323  A15     : 0x0000abab  SAR     : 0x00000000  EXCCAUSE: 0x00000005
↳0x00000005
EXCVADDR: 0x00000000  LBEG    : 0x00000000  LEND    : 0x00000000  LCOUNT : 0x00000000
↳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);
    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_
    ↪0);
}
```

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

```
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      : ↪
    ↪0x3ffb3400
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x80082580  A9      : ↪
    ↪0x3ffb0730
A10     : 0x00000004  A11     : 0x00000004  A12     : 0x00000004  A13     : ↪
    ↪0x00000000
```

(continues on next page)

(continued from previous page)

```

A14      : 0x00000000  A15      : 0x3ffb3400  SAR      : 0x0000001c  EXCCAUSE:↵
↵0x00000005
EXCVADDR: 0x00000000  LBEG     : 0x00000000  LEND     : 0x00000000  LCOUNT  :↵
↵0x00000000
Core 0 was running in ISR context:
EPC1     : 0x400d1097  EPC2     : 0x00000000  EPC3     : 0x00000000  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 Though it might seem that interrupt watchdog issues are straightforward to analyze, they often occur alongside FreeRTOS in actual applications. Many FreeRTOS operations require interrupt handling. If FreeRTOS encounters an exception, it can get stuck, appearing as an interrupt watchdog issue. Here's an example of a real crash case:

```

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
A10     : 0x3ffd4ff0  A11     : 0x0000000f  A12     : 0x00060b20  A13     :↵
↵0x00000001
A14     : 0x3ffeddf0  A15     : 0x3ffea57c  SAR     : 0x00000018  EXCCAUSE:↵
↵0x00000005
EXCVADDR: 0x00000000  LBEG    : 0x400014fd  LEND    : 0x4000150d  LCOUNT :↵
↵0xffffffffe

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->
↳xItemValue <= 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.

```

void vTaskPlaceOnEventList( List_t * const pxEventList, const TickType_t_
↳xTicksToWait )
{
    configASSERT( pxEventList );
    taskENTER_CRITICAL(&xTaskQueueMutex);

    vListInsert( pxEventList, &( pxCurrentTCB[xPortGetCoreID()]->xEventListItem )_
↳);

    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.

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: Core  0 panic'ed (LoadProhibited). Exception was unhandled.

Core  0 register dump:
PC      : 0x42016d2f  PS      : 0x00060c30  A0      : 0x820088ac  A1      : ↵
↳0x3fc98d00
A2      : 0x00000000  A3      : 0x3fc98f34  A4      : 0x0000000a  A5      : ↵
↳0x3fc98d20
A6      : 0x3fc98d00  A7      : 0x0000000c  A8      : 0x8200ca00  A9      : ↵
↳0x3fc989b0
A10     : 0x00000002  A11     : 0xffffffff  A12     : 0x00000002  A13     : ↵
↳0x3fc98bc0
A14     : 0x3fc989c0  A15     : 0x00000001  SAR     : 0x00000019  EXCCAUSE: ↵
↳0x0000001c
EXCVADDR: 0x00000008  LBEG    : 0x400556d5  LEND    : 0x400556e5  LCOUNT  : ↵
↳0xffffffff9

Backtrace: 0x42016d2c:0x3fc98d00 0x420088a9:0x3fc98d20 0x420180ff:0x3fc98d40 ↵
↳0x4037a2dd:0x3fc98d70
```

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:

```

xtensa-esp32s3-elf-addr2line -pfiaC -e ./build/idf_debug_method.elf
↳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

```

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 (LoadProhibited). Exception was unhandled.

Core  0 register dump:
PC      : 0x42016d2f  PS      : 0x00060c30  A0      : 0x820088ac  A1      :
↳0x3fc98d00
0x42016d2f: new_monkey_born at /home/libo/test_github/idf_debug_method/main/idf_
↳debug_method.c:19

A2      : 0x00000000  A3      : 0x3fc98f34  A4      : 0x0000000a  A5      :
↳0x3fc98d20
A6      : 0x3fc98d00  A7      : 0x0000000c  A8      : 0x8200ca00  A9      :
↳0x3fc989b0
A10     : 0x00000002  A11     : 0xffffffff  A12     : 0x00000002  A13     :
↳0x3fc98bc0
A14     : 0x3fc989c0  A15     : 0x00000001  SAR     : 0x00000019  EXCCAUSE:
↳0x0000001c
EXCVADDR: 0x00000008  LBEG    : 0x400556d5  LEND    : 0x400556e5  LCOUNT :
↳0xffffffff9
0x400556d5: 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
TP     : 0x3fc870f8  T0      : 0x4005890e  T1      : 0x3fc8fb2c  T2      :
↳0x00000000
S0/FP  : 0x0000000a  S1      : 0x3fc90948  A0      : 0x00000000  A1      :
↳0x3fc8fb08
A2     : 0x00000000  A3      : 0x00000001  A4      : 0x00000000  A5      :
↳0x00000009
A6     : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      :
↳0x00000000
S4     : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      :
↳0x00000000
S8     : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     :
↳0x00000000
T3     : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      :
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000005  MTVAL   :
↳0x00000008
MHARTID : 0x00000000

Stack memory:
3fc8fed0: 0x3c021bd0 0x00000000 0x3c022000 0x42015de4 0x00000000 0x00001388
↳0x00000001 0x00000000
3fc8fef0: 0x00000000 0x00000000 0x00000000 0x40384538 0x00000000 0x00000000
↳0x00000000 0x00000000
```

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:

```
❏ riscv32-esp-elf-addr2line -pfiaC -e ./build/idf_debug_method.elf 0x42007988
↳0x42007a4e
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
```

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
↳unhandled.

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
```

(continues on next page)

(continued from previous page)

```

TP      : 0x3fc870f8  T0      : 0x4005890e  T1      : 0x3fc8fb2c  T2      : _
↳0x00000000
0x4005890e: memset in ROM

S0/FP   : 0x0000000a  S1      : 0x3fc90948  A0      : 0x00000000  A1      : _
↳0x3fc8fb08
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000000  A5      : _
↳0x00000009
A6      : 0x60023000  A7      : 0x0000000a  S2      : 0x00000000  S3      : _
↳0x00000000
S4      : 0x00000000  S5      : 0x00000000  S6      : 0x00000000  S7      : _
↳0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : _
↳0x00000000
T3      : 0x00000000  T4      : 0x00000000  T5      : 0x00000000  T6      : _
↳0x00000000
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000005  MTVAL   : _
↳0x00000008
0x40380001: _vector_table at ???

MHARTID : 0x00000000

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/FreeRTOS-
↳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](#).

Table 2: Xtensa and RISC-V Exception Mapping

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

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!

=====
===== ESP32 CORE DUMP START =====
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'

===== CURRENT THREAD REGISTERS =====
exccause      0x1c (LoadProhibitedCause)
excvaddr      0x8
epc1          0x40378347
epc2          0x0
epc3          0x0
epc4          0x0
epc5          0x0
epc6          0x0
eps2          0x0
eps3          0x0
eps4          0x0
eps5          0x0
eps6          0x0
pc            0x42018677          0x42018677 <new_monkey_born+7>
lbeg          0x400556d5          1074091733
lend          0x400556e5          1074091749
lcount        0xffffffff          4294967292
sar           0x1a              26
ps            0x60c20             396320
threadptr     <unavailable>
br            <unavailable>
scompare1     <unavailable>
acclo         <unavailable>
acchi         <unavailable>
m0            <unavailable>
m1            <unavailable>
m2            <unavailable>
m3            <unavailable>
expstate      <unavailable>
f64r_lo       <unavailable>
f64r_hi       <unavailable>
f64s          <unavailable>
fcr           <unavailable>
fsr           <unavailable>
a0            0x8200a136          -2113887946
a1            0x3fc9a5a0          1070179744
a2            0x0                0
a3            0x3fc9a7ec          1070180332
a4            0x3c023088          1006776456
a5            0x3fc9a5d0          1070179792
a6            0x3fc9a5b0          1070179760
a7            0xc                12
a8            0x8200e2b8          -2113871176
a9            0x3fc9a250          1070178896
a10           0x10                16
a11           0xffffffff          -1
a12           0x10                16

```

(continues on next page)

(continued from previous page)

```

a13          0x3fc9a460          1070179424
a14          0x3fc9a260          1070178912
a15          0x1                1

===== CURRENT THREAD STACK =====
#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

===== THREADS INFO =====
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
2  process 1070182124 vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /home/
↳libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/xtensa/port.
↳c:161
3  process 1070184008 0x40378326 in esp_cpu_wait_for_intr () at /home/libo/esp/
↳github_master/components/esp_hw_support/cpu.c:145
4  process 1070169660 0x400559e0 in ?? ()
5  process 1070175732 0x400559e0 in ?? ()
6  process 1070171288 0x400559e0 in ?? ()

===== THREAD 1 (TCB: 0x3fc9a790, name: 'main') =====
#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

===== THREAD 2 (TCB: 0x3fc9aeec, name: 'IDLE0')
↳=====
#0  vPortTaskWrapper (pxCode=0x0, pvParameters=0x0) at /home/libo/esp/github_
↳master/components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:161
#1  0x40000000 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

===== THREAD 4 (TCB: 0x3fc97e3c, name: 'ipc0') =====
#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)

(continued from previous 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

===== THREAD 5 (TCB: 0x3fc995f4, name: 'esp_timer')
↳=====
#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/
↳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

===== THREAD 6 (TCB: 0x3fc98498, name: 'ipc1') =====
#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/
↳components/freertos/FreeRTOS-Kernel/portable/xtensa/port.c:532
#3  0x4037c0c5 in xTaskGenericNotifyWait (uxIndexToWait=0, ulBitsToClearOnEntry=
↳<optimized out>, ulBitsToClearOnExit=4294967295, pulNotificationValue=0x3fc98300,
↳ 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

===== ALL MEMORY REGIONS =====
Name  Address  Size  Attrs
.rtc.force_fast 0x600fe010 0x0 RW
.rtc.noinit 0x50000000 0x0 RW
.rtc.force_slow 0x50000000 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

```

(continues on next page)

(continued from previous page)

```
.iram0.bss 0x40381a00 0x0 RW
.dram0.heap_start 0x3fc96a28 0x0 RW
.coredump.tasks.data 0x3fc9a790 0x154 RW
.coredump.tasks.data 0x3fc9a4e0 0x2a0 RW
.coredump.tasks.data 0x3fc9aee0 0x154 RW
.coredump.tasks.data 0x3fc9ace0 0x200 RW
.coredump.tasks.data 0x3fc9b648 0x154 RW
.coredump.tasks.data 0x3fc9b3c0 0x280 RW
.coredump.tasks.data 0x3fc97e3c 0x154 RW
.coredump.tasks.data 0x3fc97b90 0x2a0 RW
.coredump.tasks.data 0x3fc995f4 0x154 RW
.coredump.tasks.data 0x3fc99350 0x290 RW
.coredump.tasks.data 0x3fc98498 0x154 RW
.coredump.tasks.data 0x3fc981f0 0x2a0 RW

===== ESP32 CORE DUMP END =====
=====
Done!
Coredump checksum='7b5945fe'
I (1690) esp_core_dump_uart: Core dump has been written to uart.
```

Xtensa Core Dump typically looks like the following:

```
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!

=====
===== ESP32 CORE DUMP START =====

Crashed task handle: 0x3fc91594, name: 'main', GDB name: 'process 1070142868'

===== CURRENT THREAD REGISTERS =====
ra          0x42009404      0x42009404 <app_main+56>
sp          0x3fc91510      0x3fc91510
gp          0x3fc8b400      0x3fc8b400 <__c.29+52>
tp          0x3fc88368      0x3fc88368
t0          0x4005890e      1074104590
t1          0x3fc9116c      1070141804
t2          0x0          0
fp          0x3c022000      0x3c022000
s1          0x3fc91f98      1070145432
a0          0x0          0
a1          0x3fc91148      1070141768
a2          0x0          0
a3          0x1          1
a4          0x0          0
a5          0x0          0
a6          0x60023000      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
```

(continues on next page)

(continued from previous page)

```

s11          0x0      0
t3           0x0      0
t4           0x0      0
t5           0x0      0
t6           0x0      0
pc           0x4200939e      0x4200939e <new_monkey_born+4>

===== CURRENT THREAD STACK =====
#0  new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_github/idf_debug_method/
↳main/idf_debug_method.c:21
#1  0x42009404 in app_main () at /home/libo/test_github/idf_debug_method/main/idf_
↳debug_method.c:55
#2  0x42017976 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  0x403845bc in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
↳<optimized out>) at /home/libo/esp/github_master/components/freertos/FreeRTOS-
↳Kernel/portable/riscv/port.c:234

===== THREADS INFO =====
Id  Target Id          Frame
* 1  process 1070142868 new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_
↳github/idf_debug_method/main/idf_debug_method.c:21
2  process 1070144752 vPortTaskWrapper (pxCode=0x40384f20 <prvIdleTask>,
↳pvParameters=0x0) at /home/libo/esp/github_master/components/freertos/FreeRTOS-
↳Kernel/portable/riscv/port.c:230
3  process 1070138360 0x40384796 in vPortClearInterruptMaskFromISR (prev_int_
↳level=1) at /home/libo/esp/github_master/components/freertos/FreeRTOS-Kernel/
↳portable/riscv/port.c:417

===== THREAD 1 (TCB: 0x3fc91594, name: 'main') =====
#0  new_monkey_born (zoo=zoo@entry=0x0) at /home/libo/test_github/idf_debug_method/
↳main/idf_debug_method.c:21
#1  0x42009404 in app_main () at /home/libo/test_github/idf_debug_method/main/idf_
↳debug_method.c:55
#2  0x42017976 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  0x403845bc in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
↳<optimized out>) at /home/libo/esp/github_master/components/freertos/FreeRTOS-
↳Kernel/portable/riscv/port.c:234

===== THREAD 2 (TCB: 0x3fc91cf0, name: 'IDLE') =====
#0  vPortTaskWrapper (pxCode=0x40384f20 <prvIdleTask>, pvParameters=0x0) at /home/
↳libo/esp/github_master/components/freertos/FreeRTOS-Kernel/portable/riscv/port.
↳c:230
#1  0x00000000 in ?? ()
Backtrace stopped: frame did not save the PC

===== THREAD 3 (TCB: 0x3fc903f8, name: 'esp_timer')_
↳=====
#0  0x40384796 in vPortClearInterruptMaskFromISR (prev_int_level=1) at /home/libo/
↳esp/github_master/components/freertos/FreeRTOS-Kernel/portable/riscv/port.c:417
#1  0x403847f8 in vPortExitCritical () at /home/libo/esp/github_master/components/
↳freertos/FreeRTOS-Kernel/portable/riscv/port.c:517
#2  0x40385cfa in ulTaskGenericNotifyTake (uxIndexToWait=uxIndexToWait@entry=0,
↳xClearCountOnExit=xClearCountOnExit@entry=1,
↳xTicksToWait=xTicksToWait@entry=4294967295) at /home/libo/esp/github_master/
↳components/freertos/FreeRTOS-Kernel/tasks.c:5565
#3  0x420046b6 in timer_task (arg=<error reading variable: value has been_
↳optimized out>) at /home/libo/esp/github_master/components/esp_timer/src/esp_
↳timer.c:477

```

(continues on next page)

(continued from previous page)

```

#4 0x403845bc in vPortTaskWrapper (pxCode=<optimized out>, pvParameters=
↳<optimized out>) at /home/libo/esp/github_master/components/freertos/FreeRTOS-
↳Kernel/portable/riscv/port.c:234

===== ALL MEMORY REGIONS =====
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

===== ESP32 CORE DUMP END =====
=====
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 Trace: 3 records (100 capacity) =====
36 bytes (@ 0x3fc91574, Internal) allocated CPU 0 ccount 0x02cf6d38 caller
36 bytes (@ 0x3fc9159c, Internal) allocated CPU 0 ccount 0x02cf72cc caller
===== Heap Trace Summary =====
Mode: Heap Trace Leaks
72 bytes 'leaked' in trace (2 allocations)
records: 2 (100 capacity, 3 high water mark)
total allocations: 3
total frees: 1
=====

```

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 `CONFIG_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      : 0x00000000  A4      : 0x000003e8  A5      : 
↳: 0x3fc9b75c
A6      : 0x00000000  A7      : 0x0000003e  A8      : 0x8200333d  A9      : 
↳: 0x3fc99ee0
A10     : 0x00000400  A11     : 0x00060c20  A12     : 0x00000000  A13     : 
↳: 0x00060c23
A14     : 0xb33fffff  A15     : 0xb33fffff  SAR     : 0x00000004  _
↳EXCCAUSE: 0x00000001
EXCVADDR: 0x00000000  LBEG    : 0x400570e8  LEND    : 0x400570f3  LCOUNT _
↳: 0x00000002

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");
}
```

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:

```
/home/user/github/esp-idf/rele5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c: In function 'vulnerableFunction':
/home/user/github/esp-idf/rele5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c:20:23: warning: iteration 5 invokes undefined behavior [-
↪Waggressive-loop-optimizations]
 20 |         localArray[i] = i;
    |         ~~~~~^~
/home/user/github/esp-idf/rele5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c:19:23: note: within this loop
 19 |         for (int i = 0; i <= 5; ++i) {
```

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:

```
/home/user/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c: In function 'app_main':
/home/user/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c:32:5: warning: pointer 'number' used after 'free' [-Wuse-
↪after-free]
 32 |     printf("Value after freeing: %d\n", *number);
    |     ^~~~~~
/home/zhengzhong/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/
↪main/hello_world_main.c:28:5: note: call to 'free' here
 28 |     free(number); // Free the allocated memory
```

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:


```

/home/user/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c: In function 'app_main':
/home/user/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c:21:5: error: 'number' is used uninitialized [-
↪Werror=uninitialized]
 21 |         printf("Value: %d\n", *number);
    |         ^~~~~~
/home/user/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/main/
↪hello_world_main.c:17:10: note: 'number' was declared here
 17 |         int *number; // Pointer declared but not initialized
    |         ^~~~~~
cc1: some warnings being treated as errors

```

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:

```

/home/zhengzhong/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/
↪main/hello_world_main.c: In function 'app_main':
/home/zhengzhong/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/
↪main/hello_world_main.c:29:9: warning: pointer 'data' used after 'free' [-Wuse-
↪after-free]
 29 |         free(data); // This is incorrect and may lead to undefined behavior
    |         ^~~~~~
/home/zhengzhong/github/esp-idf/rel5.1/esp-idf/examples/get-started/hello_world/
↪main/hello_world_main.c:26:9: note: call to 'free' here
 26 |         free(data);
    |         ^~~~~~

```

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

```

(continues on next page)

(continued from previous page)

```

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

TP      : 0x3fc87110  T0      : 0x37363534  T1      : 0x7271706f  T2      : ̀
↳0x33323130
S0/FP   : 0x00000069  S1      : 0x00000001  A0      : 0x3fc8f36c  A1      : ̀
↳0x3fc8acd1
A2      : 0x00000001  A3      : 0x00000029  A4      : 0x00000001  A5      : ̀
↳0x3fc8c000
A6      : 0x7a797877  A7      : 0x76757473  S2      : 0x00000009  S3      : ̀
↳0x3fc8f49e
S4      : 0x3fc8acd0  S5      : 0x00000000  S6      : 0x00000000  S7      : ̀
↳0x00000000
S8      : 0x00000000  S9      : 0x00000000  S10     : 0x00000000  S11     : ̀
↳0x00000000
T3      : 0x6e6d6c6b  T4      : 0x6a696867  T5      : 0x66656463  T6      : ̀
↳0x62613938
MSTATUS : 0x00001881  MTVEC   : 0x40380001  MCAUSE  : 0x00000007  MTVAL   : ̀
↳0x00000000
0x40380001: _vector_table at ???

MHARTID : 0x00000000

Backtrace:

panic_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/panic.c:452
452      *((volatile int *) 0) = 0; // NOLINT(clang-analyzer-core.
↳NullDereference) should be an invalid operation on targets
#0  panic_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/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> "
↳", expr=expr@entry=0x3c0217ec "") at /home/zhengzhong/github/esp-idf/rel5.1/esp-
↳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/
↳examples/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/
↳freertos/FreeRTOS-Kernel/portable/riscv/port.c:202

```

(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 Configure the following options via *menuconfig* to 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:

```
UART_PRINT_CONTROL (BLOCK0) Set the default UART boot message output mode = Disable R/W (0b11)
```

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 Startup Power Consumption Statistics Startup time statistics of ESP32-S3 at different CPU frequencies:

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

Startup time statistics of ESP32-C6 at different CPU frequencies:

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

Note: Startup power consumption is the average current consumption during the entire startup process.

2.2.4 Template

2.3 Solution Introduction

In this section we will introduce the application solutions launched by Espressif Systems.

2.3.1 Template

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

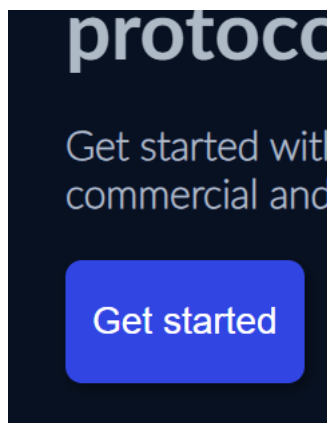
Note: This document is automatically translated using AI. Please excuse any detailed errors. The official English version is still in progress.

This article provides a detailed tutorial on how to use Wireshark to capture Wi-Fi wireless packets on Windows 10.

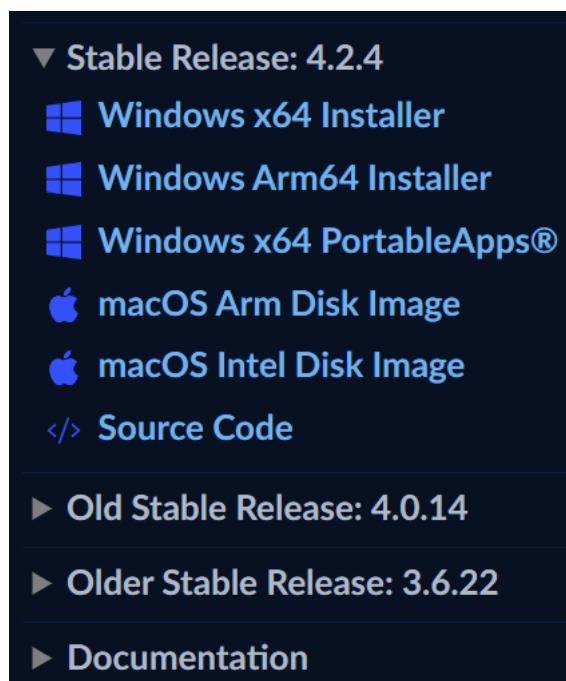
Installing Wireshark

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.

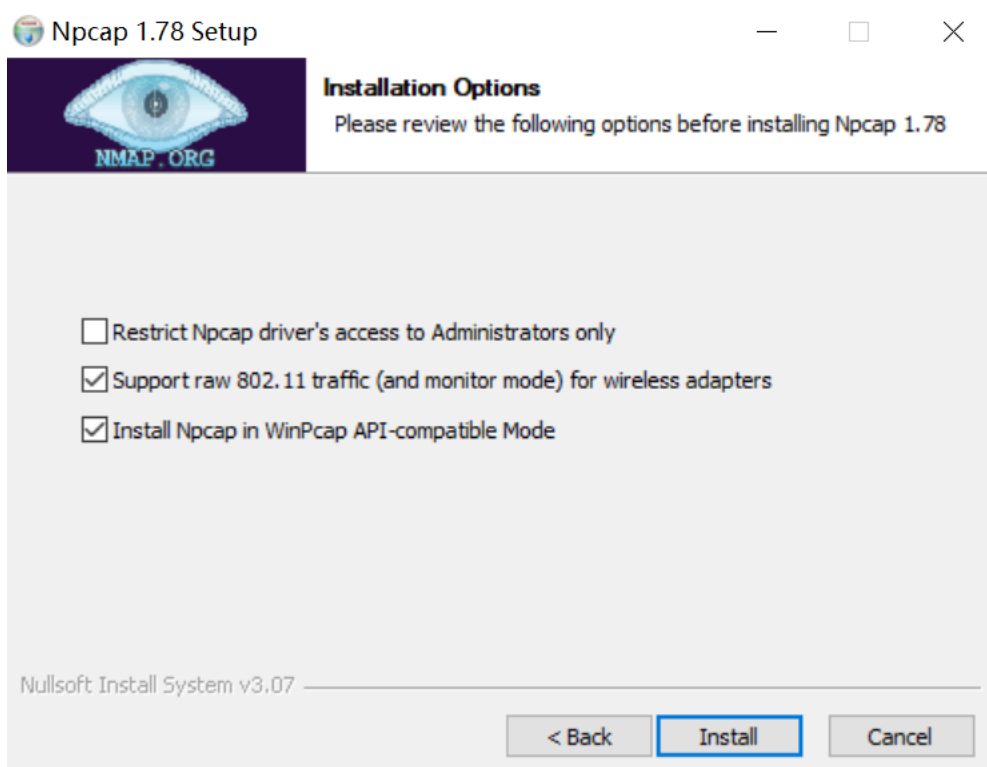
Attention: Npcap will pop up a second installation page, where you need to check the two options in the figure below (**not checked by default**), otherwise you will not be able to capture 802.11 packets.

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`:
3. To view the current mode, copy the GUID, then enter `WlanHelper.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:\>netsh wlan show interfaces

系统上有 1 个接口:

名称           : WLAN 2
描述           : Linksys WUSB600N Wireless-N USB Network Adapter with Dual-Band ver. 2
GUID           : 
物理地址       : 00:25:9c:e2:83:db
状态           : 已断开连接
无线电状态     : 硬件 开
                软件 开

承载网络状态   : 未启动
```

```
D:\>wlanhelper.exe 00971d40-8f3c-4
master, managed, monitor
```

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.











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:

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.

```
D:\>wlanhelper.exe 00971d40-8f3c-4
Success
```

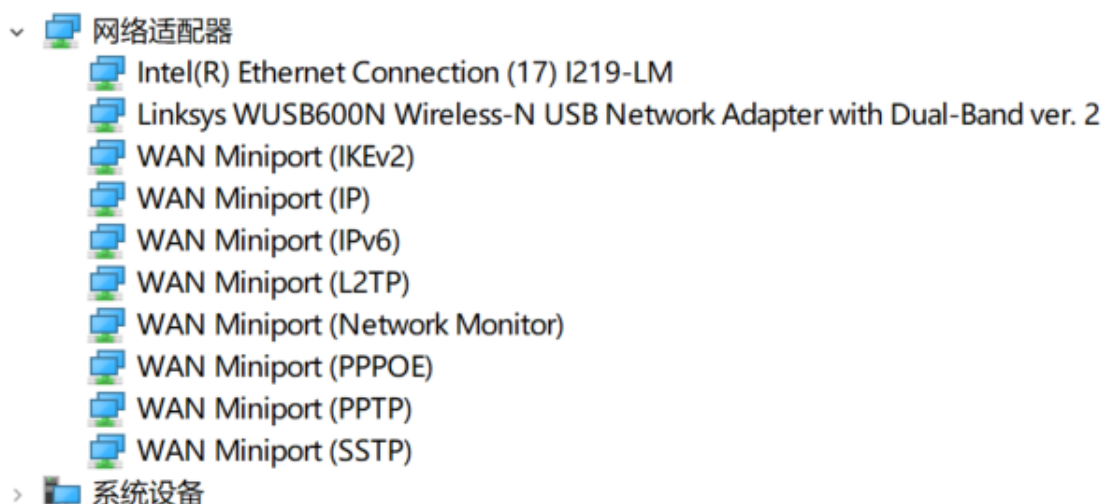

 <p>RT3070L芯片 免驱型USB无线网卡</p> <p>Linux系统上支持AP发射WiFi, 可用于WiFi实验</p> <p>接收/发射功能 网速稳定 赠送资料+视频教程</p> <p>¥38.88</p> <p>RT3070L网卡无线usb接收器kali linux cdlinux虚拟机抓包linux RT3070L白色款</p> <p>81条评价</p> <p>安翼科技小店</p> <p>【免邮】</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>	 <p>母亲节</p> <p>五仓发货 30天价保 买贵退差 技术指导 安装无忧</p> <p>¥86.00</p> <p>杰奇omnipeek无线抓包网卡\空口抓包 空中SparkLAN usb3.0双频千兆uos5g 黑</p> <p>100+条评价</p> <p>智语外设产品专营店</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>	 <p>母亲节</p> <p>五仓发货 30天价保 买贵退差 技术指导 安装无忧</p> <p>¥125.00</p> <p>EDIMAX usb无线网卡wifi接收器发射器 win10免驱ubuntu kali linux抓包 7822UAN</p> <p>5000+条评价</p> <p>五星店铺 退思数码专营店</p> <p>京东物流 门店有售 免邮</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>	 <p>RT3070L芯片 免驱型USB无线网卡</p> <p>Linux系统上支持AP发射WiFi, 可用于WiFi实验</p> <p>接收/发射功能 网速稳定 赠送资料+视频教程</p> <p>¥48.88</p> <p>RT3070L网卡无线usb接收器kali linux cdlinux虚拟机抓包linux RT3070L黑色款</p> <p>81条评价</p> <p>安翼科技小店</p> <p>【免邮】</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>	 <p>WiFi6无线网卡5G高速上网 双频2.4G/5.8G</p> <p>支持11ax高速网络</p> <p>支持游戏/直播/视频会议</p> <p>Aircrack-ng/evil-win10等工具</p> <p>赠送资料+视频教程+手机+打印</p> <p>支持拆包+高速上网 一机多用 支持开票 品质保证 质保一年</p> <p>¥98.88</p> <p>磊特wifi6无线网卡kali抓包实验测试 airmon-ng模式ax协议双频5g WiFi6-kali</p> <p>26条评价</p> <p>安翼科技小店</p> <p>【免邮】</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>
 <p>提供抓包专用驱动 双频2.4GHz&5.8GHz 支持802.11ac/b/g/n/a/a</p> <p>数据分析 空口抓包</p> <p>快! 抓包快</p> <p>软件中提供 抓包教程电脑上 上网使用 赠送资料+教程</p> <p>可开票 赠送软件</p> <p>关注店铺 急速发货</p> <p>¥168.00</p> <p>omnipeek无线抓包网卡空口抓包空中 SparkLAN usb3.0双频千兆uos5g 黑色支</p> <p>3条评价</p> <p>京仁堂农资专营店</p> <p>【满199-5】</p>	 <p>双频并发 5.8G高速上网 2.4G/5.8G双频随心切换</p> <p>5.8GHz频段</p> <p>具有比于抗能力 传输性能稳定 上网速度快</p> <p>支持kali linux, Ubuntu, Win11, Win10, win7等系统</p> <p>¥90.00</p> <p>rt8812au 高功率usb无线网卡5g双频kali linux抓包ubuntu debain rt8812au-不带延</p> <p>57条评价</p> <p>安翼科技小店</p> <p>【免邮】</p>	 <p>RT3070L芯片 免驱型USB无线网卡</p> <p>Linux系统上支持AP发射WiFi, 可用于WiFi实验</p> <p>接收/发射功能 网速稳定 赠送资料+视频教程</p> <p>¥49.88</p> <p>RT3070L网卡无线usb接收器kali linux cdlinux虚拟机抓包linux RT3070L白色外置</p> <p>81条评价</p> <p>安翼科技小店</p> <p>【免邮】</p>	 <p>母亲节</p> <p>急速发货 30天价保 买贵退差 技术指导 安装无忧</p> <p>¥70.00</p> <p>EDIMAX usb无线网卡wifi接收器发射器 win10免驱ubuntu kali linux抓包 7811UN</p> <p>5000+条评价</p> <p>五星店铺 退思数码专营店</p> <p>门店有售 免邮</p> <p><input type="checkbox"/> 对比 <input type="checkbox"/> 关注 <input type="button" value="加入购物车"/></p>	 <p>双频并发 5.8G高速上网 2.4G/5.8G双频随心切换</p> <p>5.8GHz频段</p> <p>具有比于抗能力 传输性能稳定 上网速度快</p> <p>支持kali linux, Ubuntu, Win11, Win10, win7等系统</p> <p>¥98.00</p> <p>rt8812au 高功率usb无线网卡5g双频kali linux抓包ubuntu debain rt8812au-</p> <p>57条评价</p> <p>安翼科技小店</p> <p>【免邮】</p>

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
 - 802.11n: 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



- 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:



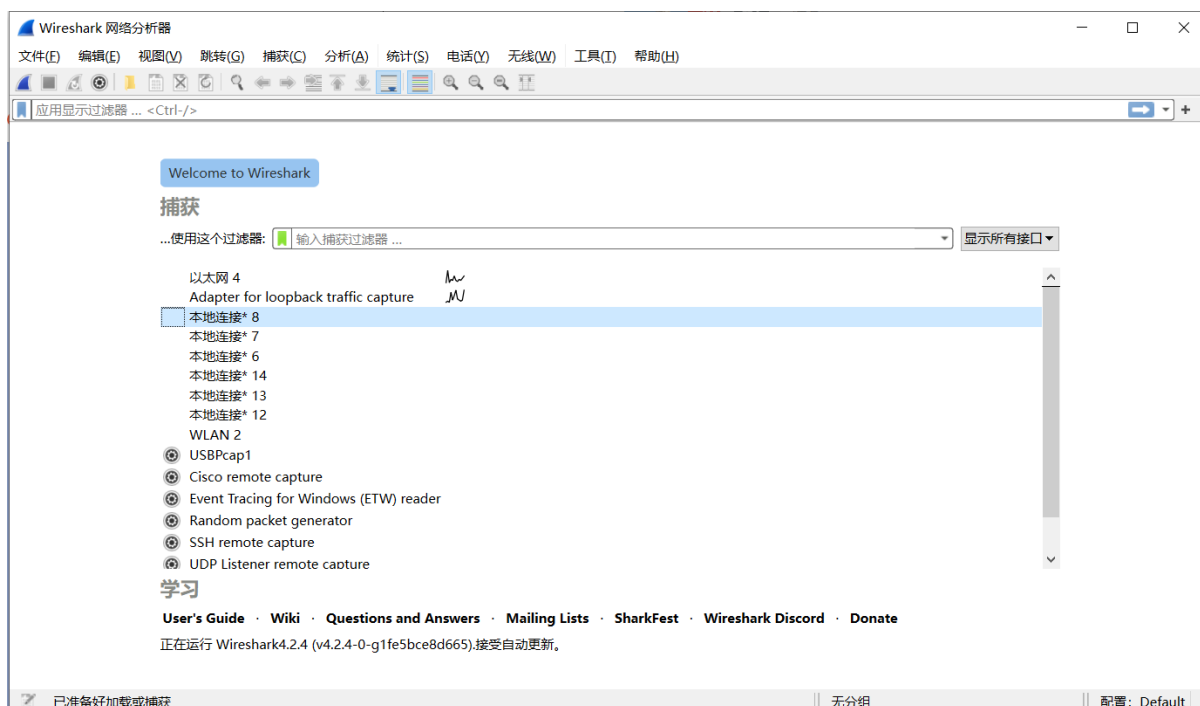
Capture Packets Using Wireshark

After installing Wireshark, you can run it to capture packets. Start Wireshark:

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](#).

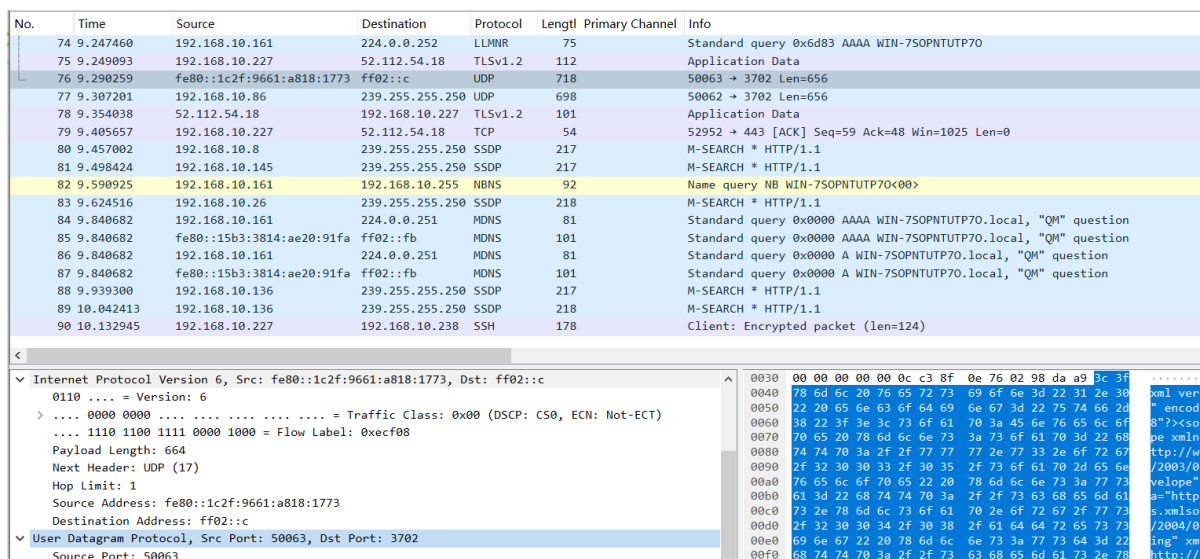
Capture Ethernet Packets You can use Wireshark to capture packets from your computer's network card and analyze the captured packets.

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.

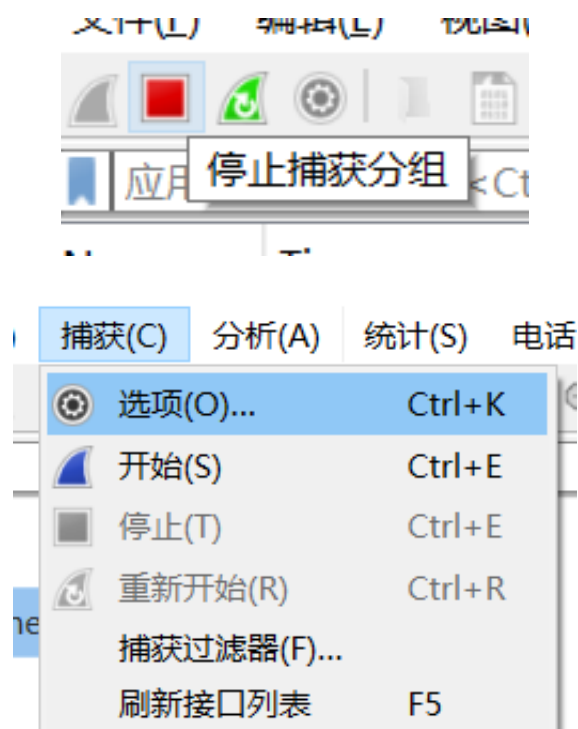
Select the Ethernet 4 interface to capture packets. Click on Start Capture at the top left corner or double-click on the interface name to begin capturing network data. While browsing the web on the local computer, Wireshark will capture data on the Local Connection interface.



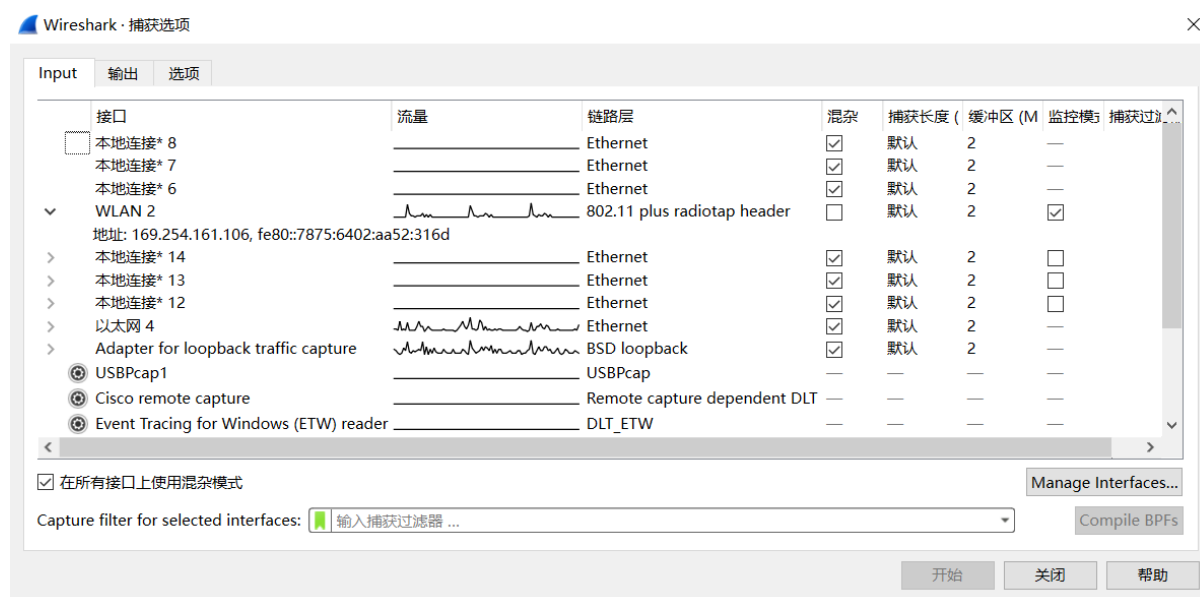
Wireshark will continue to capture data on the Local Connection . If you no longer need to capture, you can click on the Stop Capture button at the top left corner to stop capturing.

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:



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:



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:

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.

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.

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.

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.

```
D:\>ipconfig

Windows IP 配置

以太网适配器 以太网 4:

    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::4994:7891:6d3d:72ae%4
    IPv4 地址 . . . . . : 192.168.10.227
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.10.1

无线局域网适配器 本地连接* 12:

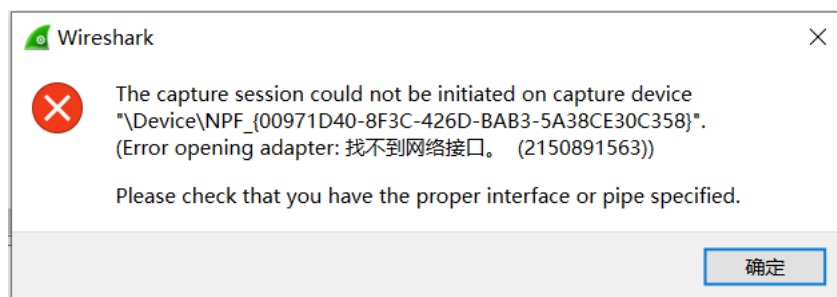
    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 13:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 WLAN 2:

    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址. . . . . : fe80::7875:6402:aa52:316d%2
    自动配置 IPv4 地址 . . . . . : 169.254.161.106
    子网掩码 . . . . . : 255.255.0.0
    默认网关. . . . . :
```

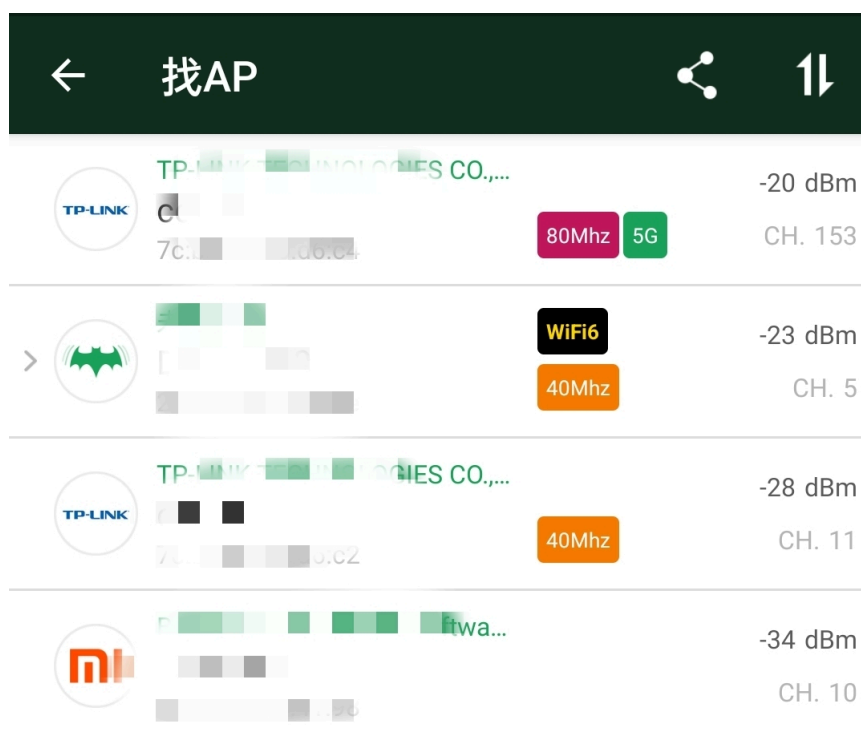


```
D:\>netsh wlan show interfaces

系统上有 1 个接口:

名称           : WLAN 2
描述           : Linksys WUSB600N Wireless-N USB Network Adapter with Dual-Band ver. 2
GUID           : 00971d40-8f3c-426d-
物理地址       : 00:25:9c:e2:
状态           : 已连接
SSID           :
BSSID          : 70:3a:73:84:
网络类型       : 结构
无线电类型     : 802.11n
身份验证       : WPA2 - 个人
密码           : CCMP
连接模式       : 配置文件
信道           : 60
接收速率 (Mbps) : 144
传输速率 (Mbps) : 144
信号           : 80%
配置文件       :

承载网络状态   : 未启动
```



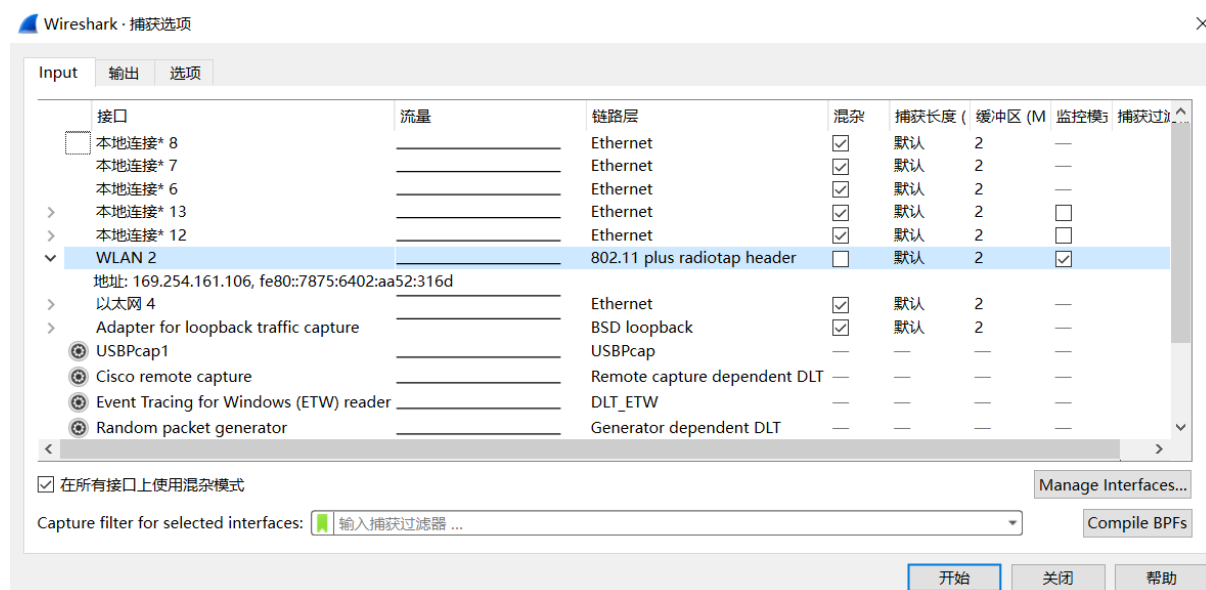
To enable the monitor mode, enter `WlanHelper.exe + GUID + mode monitor`, for example:

```
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

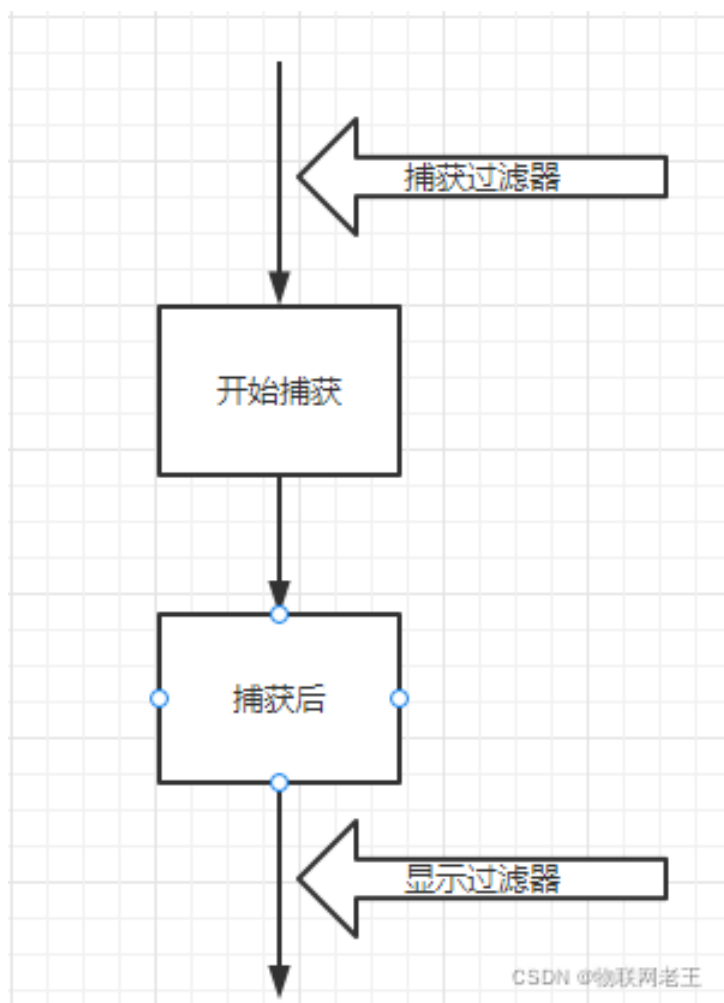
Wireshark has set up two filters: capture filter and display filter.

- **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;

- **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.



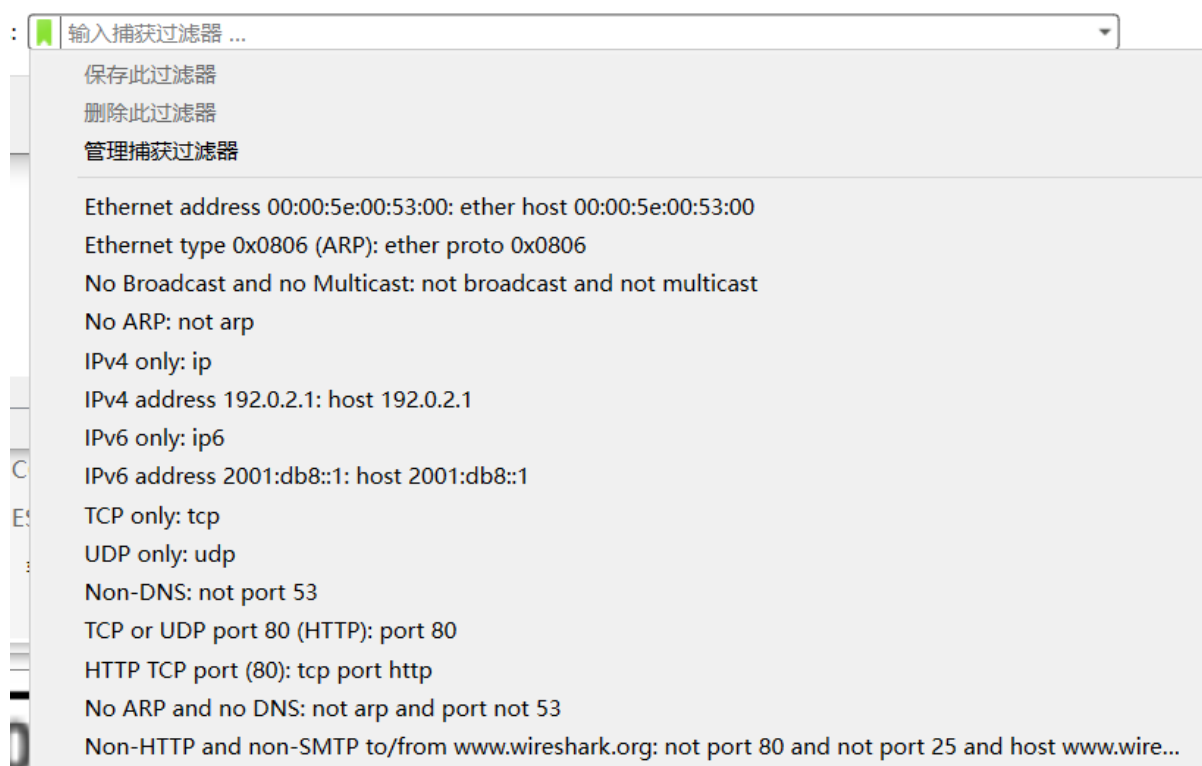
[Image source](#)

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:



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.



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
IPv4 address 192.0.2.1	host 192.0.2.1
IPv6 only	ip6
IPv6 address 2001:db8::1	host 2001:db8::1
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
Non-HTTP and non-SMTP to/from www.wireshark.org	not port 80 and not port 25 and host www.wireshark.org

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 + ...
2. Primitive = id + Modifier 1 + Modifier 2 + ...
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 `snup` and belong to FTP ports or data:

```
gateway snup and (port ftp or ftp-data)
```

This example captures no ARP type packets:

```
not arp
```

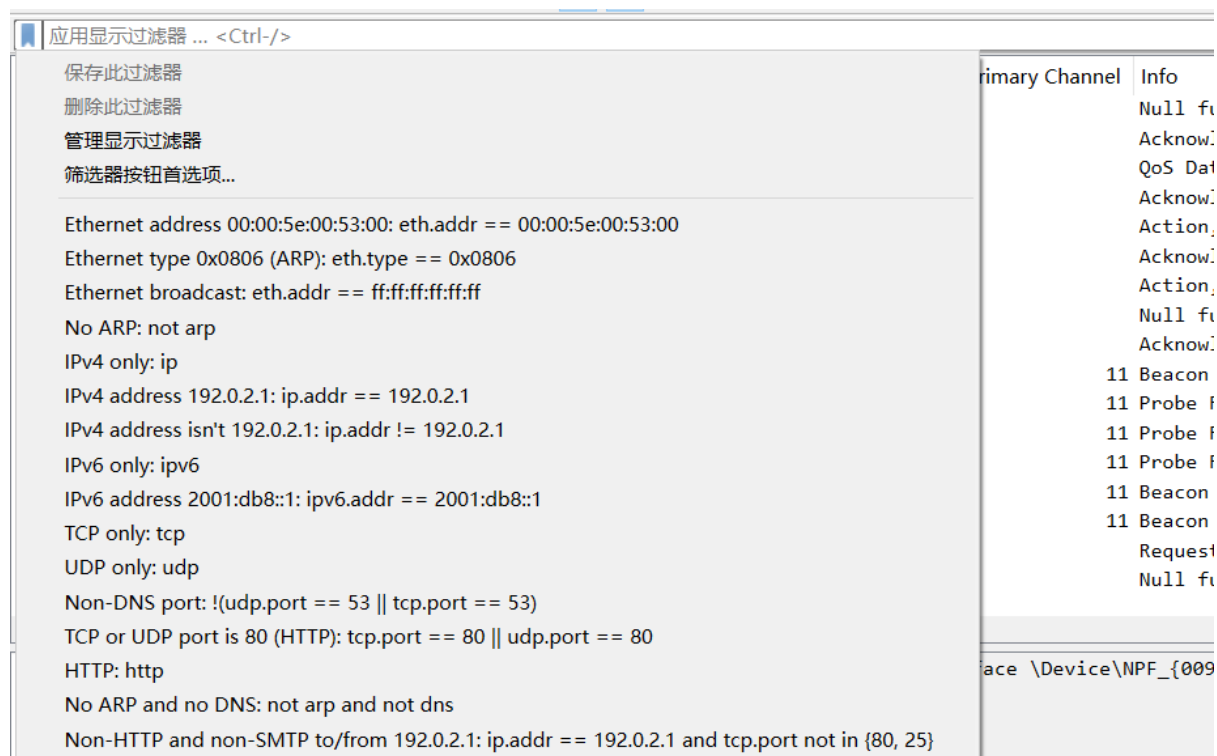
This example captures only TCP or UDP type packets:

```
tcp || udp
```

It's worth mentioning that this filter comes with syntax checking, and the green frame at the bottom indicates correct syntax.

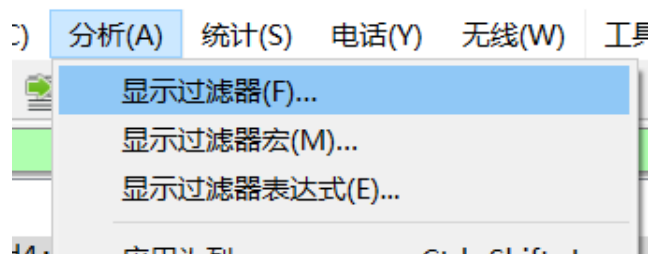
Note: For more usage and examples of capture filters, please refer to the Wireshark official explanation [capture filters wiki](#) and [capture filters Gitlab](#).

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:



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.

A reminder once again, display filter expressions and capture filter expressions should not be confused. The form of a capture filter expression is: `tcp port 80`. The three parts of the expression are explained as follows:



Wireshark · 显示过滤器

Filter 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
HTTP	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}

- 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:

Table 6.6. Display Filter comparison operators

English	Alias	C-like	Description	Example
eq	any_eq	==	Equal (any if more than one)	<code>ip.src == 10.0.0.5</code>
ne	all_ne	!=	Not equal (all if more than one)	<code>ip.src != 10.0.0.5</code>
	all_eq	===	Equal (all if more than one)	<code>ip.src === 10.0.0.5</code>
	any_ne	!==	Not equal (any if more than one)	<code>ip.src !== 10.0.0.5</code>
gt		>	Greater than	<code>frame.len > 10</code>
lt		<	Less than	<code>frame.len < 128</code>
ge		>=	Greater than or equal to	<code>frame.len ge 0x100</code>
le		<=	Less than or equal to	<code>frame.len <= 0x20</code>
contains			Protocol, field or slice contains a value	<code>sip.To contains "a1762"</code>
matches		~	Protocol or text field matches a Perl-compatible regular expression	<code>http.host matches "acme\\. (org com net)"</code>

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://ww.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 [capture filters wiki](#) and [capture filters Gitlab](#).

Table 6.7. Display Filter Logical Operations

English	C-like	Description	Example
and	&&	Logical AND	<code>ip.src==10.0.0.5 and tcp.flags.fin</code>
or		Logical OR	<code>ip.src==10.0.0.5 or ip.src==192.1.1.1</code>
xor	^^	Logical XOR	<code>tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29</code>
not	!	Logical NOT	<code>not llc</code>
[...]		Subsequence	See “Slice Operator” below.
in		Set Membership	<code>http.request.method in {"HEAD", "GET"}</code> . See “Membership Operator” below.

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 `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:

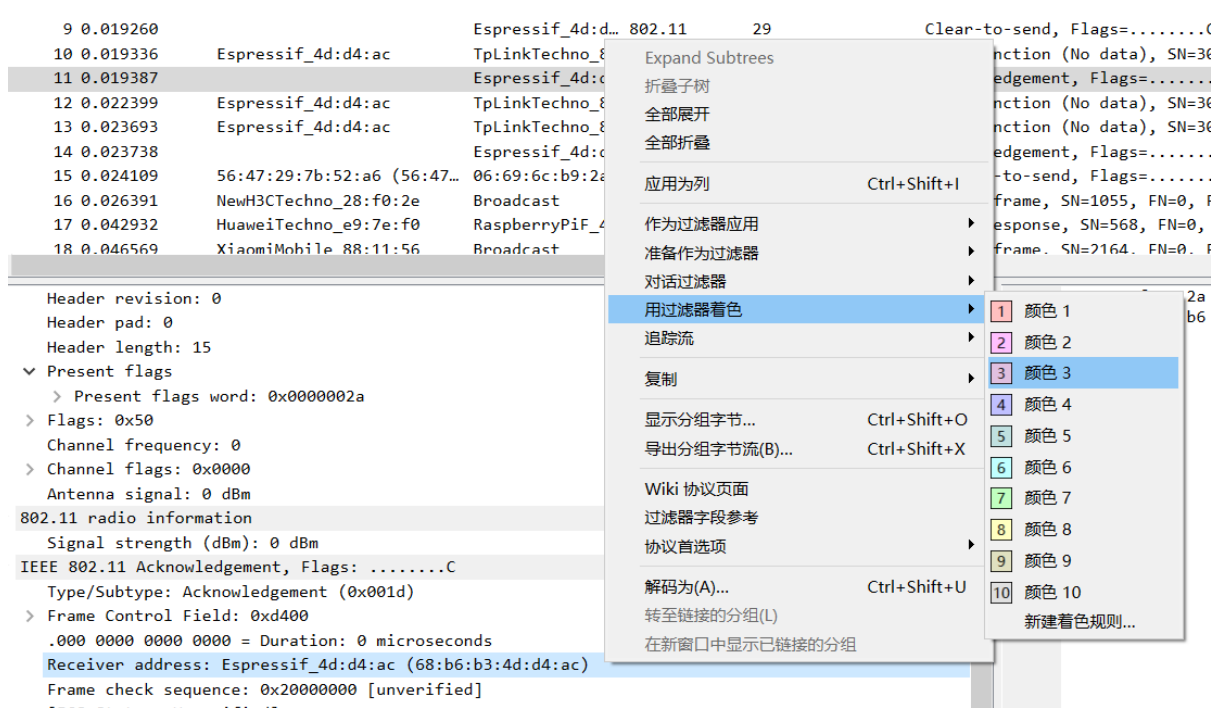
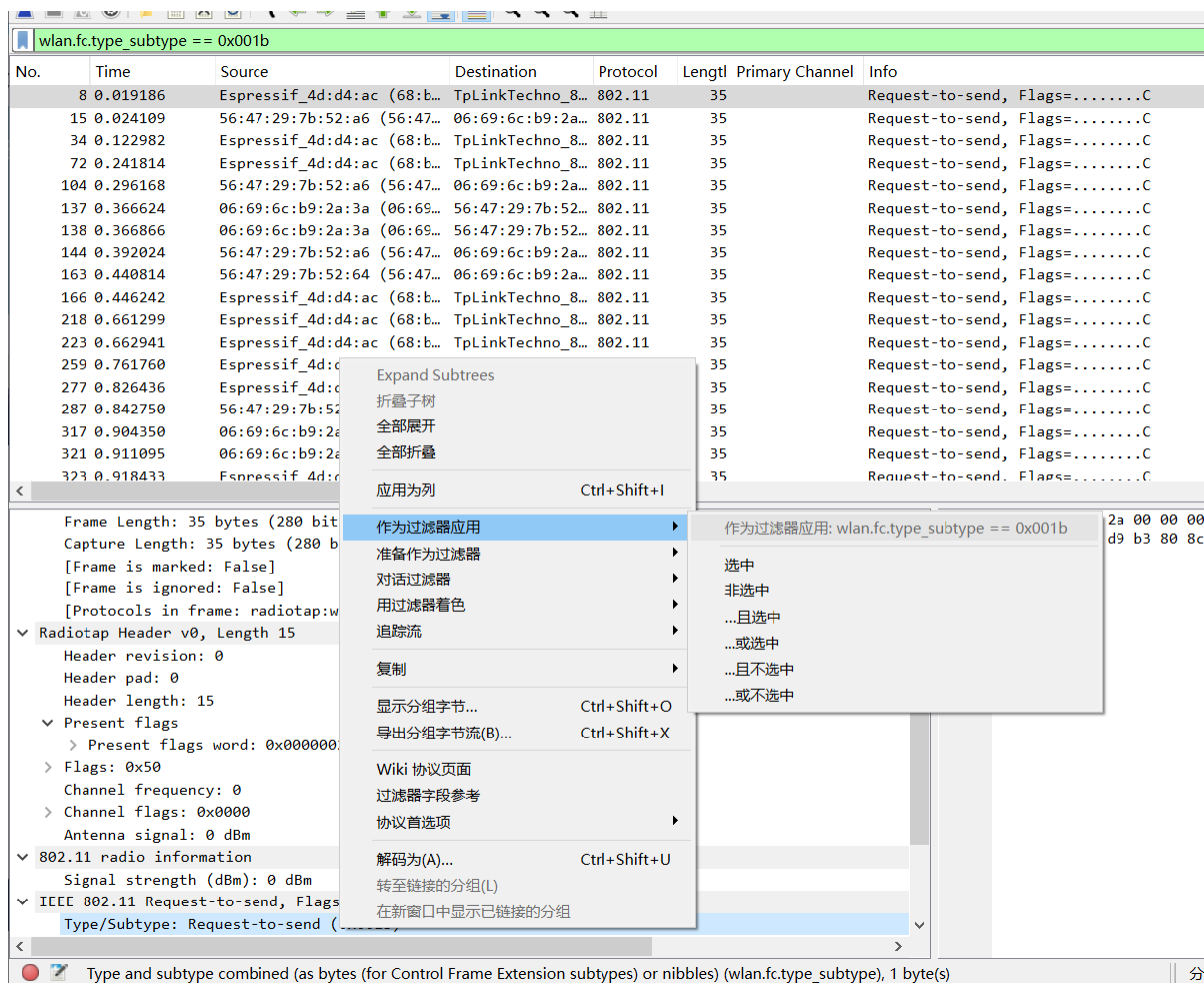
You can manage or reset existing rules in `View - Conversation Coloring`:

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:

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:



7	0.019110	00:05:0c:09:26:38 (00:05...	00:47:29:7b:52:a6	802.11	47	802.11 BULK ACK, Flags=.....C
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
10	0.019336	Espressif_4d:d4:ac	TplinkTechno_8...	802.11	43	Null function (No data), SN=3005, FN=0, Flags=...R..TC
11	0.019387		Espressif_4d:d...	802.11	29	Acknowledgement, Flags=.....C
12	0.022399	Espressif_4d:d4:ac	TplinkTechno_8...	802.11	43	Null function (No data), SN=3007, FN=0, Flags=.....TC
13	0.023693	Espressif_4d:d4:ac	TplinkTechno_8...	802.11	43	Null function (No data), SN=3008, FN=0, Flags=.....TC
14	0.023738		Espressif_4d:d...	802.11	29	Acknowledgement, Flags=.....C
15	0.024109	56:47:29:7b:52:a6 (56:47...	06:69:6c:b9:2a...	802.11	35	Request-to-send, Flags=.....C
16	0.026391	NewH3CTechno 28:f0:2e	Broadcast	802.11	240	Beacon frame. SN=1055. FN=0. Flaes=.....C. BI=100. SSID="Lenovo"™

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(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)

- ✓ 主工具栏(M)
- ✓ 过滤器工具栏(F)
- ✓ 状态栏(S)
- 全屏(F) F11
- ✓ 分组列表(L)
- ✓ 分组详情(D)
- ✓ 分组字节流(B)
- 分组图(D)
- 时间显示格式(T) ▶
- 名称解析(U) ▶
- 缩放(Z) ▶
- 展开子树(X) Shift+右方向
- 折叠子树 Shift+左方向
- 展开全部(E) Ctrl+右方向
- 收起全部(A) Ctrl+左方向
- 着色分组列表
- 着色规则(C)...
- 对话着色 ▶
 - 1 颜色 1 Ctrl+1
 - 2 颜色 2 Ctrl+2
 - 3 颜色 3 Ctrl+3
 - 4 颜色 4 Ctrl+4
 - 5 颜色 5 Ctrl+5
 - 6 颜色 6 Ctrl+6
 - 7 颜色 7 Ctrl+7
 - 8 颜色 8 Ctrl+8
 - 9 颜色 9 Ctrl+9
 - 10 颜色 10
 - 重置着色 Ctrl+空格
 - 新建着色规则...
- 重置布局 Ctrl+Shift+W
- 调整列宽 Ctrl+Shift+R
- 内部 ▶
- 在新窗口显示分组(W)
- 重新载入为文件格式/捕获 Ctrl+Shift+F
- 重新加载(R) Ctrl+R

length (dBm): 0 dBm
 Request-to-send, Flags:C
 type: Request-to-send (0x001b)
 control Field: 0xb400
 0111 1010 = Duration: 1146 microseconds
 address: 06:69:6c:b9:2a:3a (06:69:6c:b9:2a:3a)

on	Protocol	Length	Prim
	802.11	3516	
e:b5:31...	802.11	928	
a:1c:44...	802.11	990	
echno_8...	802.11	43	
9:7b:52...	802.11	47	
echno_8...	802.11	35	
if_4d:d...	802.11	29	
echno_8...	802.11	43	
if_4d:d...	802.11	29	
echno_8...	802.11	43	
echno_8...	802.11	43	
if_4d:d...	802.11	29	
c:b9:2a...	802.11	35	
st	802.11	240	
ryPiF_4...	802.11	544	
st	802.11	404	
ryPiF_4...	802.11	256	
ryPiF_4...	802.11	256	

```

0... .. = reserved: 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, Environr
  > Tag: FRP Information
    
```

No.	Time	Source	Destination	Protocol	Length	Current Channel	Info
12	0.022399	Espressif_4d:d4:ac	TpLinkTechno_8...	802.11	43		
13	0.023693	Espressif_4d:d4:ac	TpLinkTechno_8...	802.11	43		
14	0.023738		Espressif_4d:d...	802.11	29		
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	
17	0.042932	HuaweiTechno_e9:7e:f0	RaspberryPiF_4...	802.11	544	11	
18	0.046569	XiaomiMobile_88:11:56	Broadcast	802.11	404	11	
19	0.048856	UTTTechnolog_ab:db:70	RaspberryPiF_4...	802.11	256	11	
20	0.051035	UTTTechnolog_ab:db:70	RaspberryPiF_4...	802.11	256	11	
21	0.055514	XiaomiMobile_b9:6b:be	RaspberryPiF_4...	802.11	523	11	
22	0.069932	HuaweiTechno_e9:7e:f1	Broadcast	802.11	258	11	
23	0.073936	XiaomiMobile_88:11:56	96:32:53:84:d8...	802.11	484	11	
24	0.086600	XiaomiMobile_b9:6b:be	RaspberryPiF_4...	802.11	523	11	
25	0.091335	XiaomiMobile_88:11:56	96:32:53:84:d8...	802.11	484	11	
26	0.092000		00:69:6c:b9:2a...	802.11	29		
27	0.094568	TpLinkTechno_6b:d6:c2	Broadcast	802.11	247	11	
28	0.095877	53:cf:f5:55:c5:2b (53:cf...	1a:a3:df:5e:ea...	802.11	76		
29	0.104560	TpLinkTechno_80:8c:81	Broadcast	802.11	298	11	

标题: Info 类型: Information 字段: 输入字段... 发生: Resolve Names: 确定

No.	Time	Source	Destination	Protocol	Length	Current Channel	Info
12	0.022399	Espressif					
13	0.023693	Espressif					
14	0.023738						
15	0.024109	56:47:29:7b:52:a6					
16	0.026391	NewH3CTechno_28:f0:2e					
17	0.042932	HuaweiTechno_e9:7e:f0					
18	0.046569	XiaomiMobile_88:11:56					
19	0.048856	UTTTechnolog_ab:db:70					
20	0.051035	UTTTechnolog_ab:db:70					
21	0.055514	XiaomiMobile_b9:6b:be					

	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	Hu
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	Hu
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	Tp
28	2024-05-08 11:56:45.167157	5E
--	-- -- -- -- -- -- -- -- --	--