# ESP32

## Frequently Asked Questions

Version 1.1
Espressif Systems

# About This Guide

The document lists the FAQ about ESP32 and the answers.

## Release Notes

| Date | Version | Release notes |
|---|---|---|
| 2018.08 | V1.0 | Initial release. |
| 2018.10 | V1.1 | Added a new section “How can I modify the default method of RF calibration?” in Chapter 1. |

## Documentation Change Notification

**Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at** *https://www.espressif.com/en/subscribe*.

## Certification

**Download certificates for Espressif products from** *https://www.espressif.com/en/certificates*.

# Table of Contents

# 1. RF

## 1.1. RF Matching and Materials

### 1.1.1. How can I optimize the second harmonic and other spurious signals created by my own products?

The second harmonic mainly comes from the radiation generated by the RF link and PA (power amplifier) power supply. The backplane (board size) and the product also make impact on the second harmonic. Therefore, it is recommended that you:

- Add a ground capacitor of approximately 2.4 pF to the RF matching circuit to reduce the spurious radiation coming from the RF link;
- Add a series inductor to the PA power supply (Pins 3 and 4 of ESP32) to reduce the spurious radiation coming from it.

## 1.2. Wi-Fi Performance

### 1.2.1. What Wi-Fi protocols and parameters does ESP32 support?

| Standard/Speed (bps) | Standard TX EVM (db) | ESP32 TX Power (dbm) | ESP32 TX EVM (db) | ESP32 RX Sensitivity (dbm) |
|---|---|---|---|---|
| 802.11b 1M | –10 | 19.5 ± 2 dB | –17 | –98 |
| 802.11b 2M | –10 | 19.5 ± 2 dB | –17 | –96 |
| 802.11b 5.5M | –10 | 19.5 ± 2 dB | –17 | –94 |
| 802.11b 11M | –10 | 19.5 ± 2 dB | –17 | –91 |
| 802.11g 6M | –5 | 18 ± 2 dB | –19 | –93 |
| 802.11g 9M | –8 | 18 ± 2 dB | –19 | –92 |
| 802.11g 12M | –10 | 18 ± 2 dB | –19 | –90 |
| 802.11g 18M | –13 | 18 ± 2 dB | –20 | –88 |
| 802.11g 24M | –16 | 16.5 ± 2 dB | –20 | –85 |
| 802.11g 36M | –19 | 16.5 ± 2 dB | –24 | –82 |
| 802.11g 48M | –22 | 15 ± 2 dB | –26 | –78 |
| 802.11g 54M | –25 | 14 ± 2 dB | –27 | –74 |
| 802.11n HT20 MCS0/6.5M/7.2M | –5 | 18 ± 2 dB | –19 | –90 |
| 802.11n HT20 MCS1/13M/14.4M | –10 | 18 ± 2 dB | –19 | –90 |

| | | | | |
|---|---|---|---|---|
| 802.11n HT20 MCS2/19.5M/21.7M | –13 | 18 ± 2 dB | –21 | –87 |
| 802.11n HT20 MCS3/26M/28.9M | –16 | 16.5 ± 2 dB | –20 | –84 |
| 802.11n HT20 MCS4/39M/43M | –19 | 16.5 ± 2 dB | –23 | –81 |
| 802.11n HT20 MCS5/52M/57.8M | –22 | 15 ± 2 dB | –26 | –77 |
| 802.11n HT20 MCS6/58.5M/65M | –25 | 14 ± 2 dB | –27 | –75 |
| 802.11n HT20 MCS7/65M/72.2M | –27 | 13 ± 2 dB | –29 | –71 |
| 802.11n HT40 MCS0/6.5M/7.2M | –5 | 18 ± 2 dB | –19 | –89 |
| 802.11n HT40 MCS1/13M/14.4M | –10 | 18 ± 2 dB | –19 | –87 |
| 802.11n HT40 MCS2/19.5M/21.7M | –13 | 18 ± 2 dB | –21 | –84 |
| 802.11n HT40 MCS3/26M/28.9M | –16 | 16.5 ± 2 dB | –20 | –82 |
| 802.11n HT40 MCS4/39M/43M | –19 | 16.5 ± 2 dB | –23 | –79 |
| 802.11n HT40 MCS5/52M/57.8M | –22 | 15 ± 2 dB | –26 | –75 |
| 802.11n HT40 MCS6/58.5M/65M | –25 | 14 ± 2 dB | –27 | –73 |
| 802.11n HT40 MCS7/65M/72.2M | –27 | 13 ± 2 dB | –29 | –69 |

Besides, the ESP32 family also supports both BT and BLE (Bluetooth Low Energy).

### 1.2.2. How can I suppress the harmonics of 80 MHz?

If the harmonics of 80 MHz (160 MHz, 240 MHz, 320 MHz, etc) exceed the allowable levels of spurious emissions, you can add a resistor of approximately 470 Ω to the data transmission circuit (TXD) to suppress those harmonics.

### 1.2.3. How can I modify the default method of RF calibration?

- During RF initialization, the partial-calibration method is used by default for RF calibration.

  To use this method, please go to ***menuconfig*** and enable *CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE*.

- If boot duration is not critical, please use the overall-calibration method instead.

  To switch to the overall-calibration method, go to ***menuconfig*** and disable *CONFIG_ESP32_PHY_CALIBRATION_AND_DATA_STORAGE*.

- If you use the default method of RF calibration, and want to add the function of triggering overall calibration as a last-resort remedy:

  Please erase the NVS partition to trigger overall calibration.

# 2. Application

## 2.1. Functions and APIs

### 2.1.1. How to determine the maximum stack usage by the thread?

Please call Function `UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask )` to get the information on the maximum stack usage. The stack usage increases/decreases with task execution and interrupt handling. This function can return the minimum remaining stack space after the task started, which enables you to estimate the maximum stack usage. In File ***FreeRTOSConfig.h***, the Macro `INCLUDE_uxTaskGetStackHighWaterMark` must be set to 1 for this function to work. Note that this option is enabled by default. More details are in *https://www.freertos.org/uxTaskGetStackHighWaterMark.html*

# 3. Audio

## 3.1. Memory and Decoding

### 3.1.1. How can I use PSRAM and get relevant documents or demos?

To enable PSRAM, you can use `make menuconfig -> Component config -> ESP32-specific`,

- If using ESP-IDF V2.1, please choose `Capability allocator can allocate SPI RAM memory`;
- If using ESP-IDF V3.0 or later versions, please choose `Support for external, SPI-connected RAM`.

After enabling PSRAM, the option `malloc() can also allocate in SPI SRAM` can enable using malloc to dynamically allocate PSRAM. When allowed, dynamic memory allocation uses PSRAM automatically.

And the option `Always put malloc()s smaller than this size, in bytes, in internal RAM` sets the threshold value.

- If the size of the memory dynamically allocated by malloc is greater than the threshold value, then PSRAM is used. Otherwise, RAM is used.
- If the remaining RAM is not enough, the PSRAM will be used.

### 3.1.2. How can I get the remaining size of PSRAM and RAM?

| Version | Type | Sub-Type | API |
|---|---|---|---|
| ESP-IDF V3.0 & later versions | With PSRAM | All remaining memory (include PSRAM and internal RAM) | esp_get_free_heap_size() |
| | | Remaining RAM size | heap_caps_get_free_size(MALLOC_CAP_INTERNAL) |
| | | Remaining DRAM size | heap_caps_get_free_size(MALLOC_CAP_INTERNAL \| MALLOC_CAP_8BIT) |
| | Without PSRAM | Remaining RAM size | heap_caps_get_free_size(MALLOC_CAP_INTERNAL) |
| | | Remaining DRAM size | esp_get_free_heap_size() |
| ESP-IDF V2.1 | With PSRAM | All remaining memory (include PSRAM and internal RAM) | xPortGetFreeHeapSizeCaps(MALLOC_CAP_SPIRAM) |
| | | Remaining RAM size | xPortGetFreeHeapSizeCaps(MALLOC_CAP_INTERNAL) |
| | | Remaining DRAM size | xPortGetFreeHeapSizeCaps(MALLOC_CAP_INTERNAL \| MALLOC_CAP_8BIT) |
| | Without PSRAM | Remaining RAM size | esp_get_free_heap_size() |

## 3.2. Peripherals

### 3.2.1. Since ESP32 has only one CHIP_PU pin and no reset pin, can CHIP_PU also be used for reset? Or it can only work as a low-power hardware control interface? If there is no reset signal, does it mean that a watchdog or other tools can be used to ensure that the device will never crash?

CHIP_PU is the reset pin.

### 3.2.2. The pins for I2S signals are located far apart from one another in the reference designs provided by Espressif. Can these pins be located closer together? For example, configure all the I2S signals to GPIO5, GPIO18, GPIO23, GPIO19 and GPIO22; and configure all the I2C signals to GPIO25 and GPIO26, or GPIO32 and GPIO33.

All I2S I/Os can be allocated freely. Please note that some I/Os can only be used as input pins. For details, please refer to the last page of *ESP32 Datasheet*.

### 3.2.3. Can signal outputs for pulse width modulation (PWM) be configured to any pins other than flash, SD, I2S, I2C and UART?

The PWM signal outputs can be configured to any pins, except for the input-only I/Os.

# 4. AT

## 4.1. Where can I get all the resources related to ESP32 AT?

- ESP32 AT bin files: *https://www.espressif.com/en/support/download/at*
- ESP32 AT document: 中文版 | *English*

You can also develop more AT commands based on the core codes of Espressif AT commands. Please find more information on ESP32 AT demos on GitHub: *https://github.com/espressif/esp32-at*.

## 4.2. Why are AT commands keep prompting "busy"?

The processing mechanism of the AT commands is serial, i.e. one command at a time. The "busy" prompt indicates that the previous command has not been executed yet, and the system cannot respond to the current input.

Any input through serial ports is considered to be a command input, so the system will also prompt "busy" or "ERROR" when there is any extra invisible character input.

For example, when the users enter `AT+GMR (line break CR LF) + (space)` through serial ports, the system will execute the command immediately, because the `AT+GMR (line break CR LF)` is already considered to be a complete AT command.

Therefore, the `space` following the AT+GMR command will be treated as a second command. If `AT+GMR` has not been processed by the time of receiving the space, the system will prompt "busy". However, if AT+GMR has been processed, the system will prompt "ERROR", since space is an incorrect command.

## 4.3. After the BLE starts broadcasting, why some mobile phones cannot successfully scan broadcasts?

1. Firstly, please confirm whether your mobile phone supports BLE function.
2. Secondly, some mobile phones (for example, iPhones) display only Classic Bluetooth in ***"Settings"*** -> ***"Bluetooth"*** (by default), and the BLE broadcast will be filtered out by the mobile phone. It is recommended to use a dedicated BLE application to debug the BLE function. For example, LightBlue application can be used on iPhone.
3. Bluetooth specification provides the correct format of the BLE broadcast packet. Mobile phones tend to filter out packets that do not conform to the specified format and display only the correct ones. For details, please refer to the document CSS v7.

# 5. Bluetooth

## 5.1. Classic Bluetooth

### 5.1.1. What are the Classic Bluetooth profiles supported by ESP32?

- ESP-IDF V3.1: HFP Client (not HF gateway)
- ESP-IDF V3.0: A2DP Source/A2DP Sink/AVRCP/AVDTP/SPP/RFCOMM

### 5.1.2. How to connect mobile phones and play music using ESP32 Bluetooth?

ESP32 is used as A2DP receiver when connected to a cell phone to play music. Please note that the *A2DP Sink Demo* uses a mobile phone to obtain SBC encoded data stream only. In order to play sounds, you will also need peripherals, such as codec modules and codec conversion capabilities, D/A converter and speaker, etc.

### 5.1.3. What is the SPP performance of ESP32?

Using two ESP32 boards to run SPP, one-way throughput can reach up to 1900 Kbps (about 235 KB/s), which is close to the theoretical value in the specification.

### 5.1.4. What is the compatibility between ESP32 and Classic Bluetooth devices?

Tests proved that ESP32 is compatible with most of the mainstream mobile phone models from major brands, such as Apple, Huawei, Xiaomi, OPPO, Meizu, OnePlus, ZTE, 360, with only a few exceptions.

### 5.1.5. What is ESP32 Classic BT operating current?

| A2DP（CPU 160 Mhz，DFS = false，commit *a7a90f*) | | | |
|---|---|---|---|
| **Current** | **MAX (mA)** | **Min (mA)** | **Average (mA)** |
| Scanning | 106.4 | 30.8 | 37.8 |
| Sniff | 107.6 | 31.1 | 32.2 |
| Play Music | 123 | 90.1 | 100.4 |

## 5.2. Bluetooth Low Energy

### 5.2.1. What BLE profiles does ESP32 support?

At the moment, ESP32 BLE fully supports some basic profiles, such as GATT/SMP/GAP, and some self-defined profiles. The ones that have already been implemented include BLE

HID (receiving side), BLE SPP-Like, Battery, DIS, Blu-Fi (Bluetooth Network Configuration-transmitting side), and so on.

### 5.2.2. What is ESP32 BLE throughput?

ESP32's BLE throughput depends on various factors such as environmental interference, connection interval, MTU size, and the performance of peer devices. For details, please refer to ble_throughput example in IDF. The maximum throughput of BLE communication between ESP32 boards can reach up to 700 Kbps, which is about 90 KB/s.

### 5.2.3. What are the compatibility and performance of ESP32 BLE Bluetooth networking? Is it open source?

The Bluetooth networking for ESP32 is called Blu-Fi.

The Blu-Fi networking compatibility of ESP32 is consistent with its BLE compatibility. Tests proved that ESP32 is compatible with most of the mainstream mobile phone models from major brands, such as Apple, Huawei, Xiaomi, OPPO, Meizu, OnePlus, ZTE, with only a few exceptions (To achieve better results with limited-compatibility models, please lower MTU parameters. It will slightly increase the connection time).

The Blu-Fi networking can be completed in 1 to 2 seconds. At present, the Blu-Fi networking supports many features, such as WPA2-enterprise certificate transmission, connection status reporting, and encryption method selection.

Currently, the codes for Blu-Fi protocol and for the mobile phone application are not open source, but they might be released in the future.

### 5.2.4. What is ESP32 BLE operating current?

| Under CPU 160 Mhz, DFS = false; ESP-IDF V3.1 | | | |
|---|---|---|---|
| **Current** | **MAX (mA)** | **Min (mA)** | **Average (mA)** |
| Advertising：<br><br>Adv Interval = 40 ms | 142.1 | 32 | 42.67 |
| Scanning:<br><br>Scan Interval = 160 ms, Window = 20 ms | 142.1 | 32 | 44.4 |
| Connection(Slave):<br><br>Connection Interval = 20 ms, latency = 0 | 142.1 | 32 | 42.75 |
| Connection(Slave):<br><br>Connection Interval = 80 ms, latency = 0 | 142.1 | 32 | 35.33 |

## 5.3. Coexistence

### 5.3.1. How do ESP32 Bluetooth dual modes coexist? And how to use dual-mode Bluetooth?

ESP32 BT&BLE dual-mode Bluetooth is very easy to use. It does not require any special configuration or calling functions. As a developer, all you have to know is (1) BLE calls the BLE APIs, and (2) Classic Bluetooth calls the Classic Bluetooth APIs. For details, please refer to *ESP32 BT&BLE Dual-mode Bluetooth*.

### 5.3.2. How do ESP32 Bluetooth and Wi-Fi coexist?

In the ***menuconfig*** menu, there is a special option called ***"Software controls WiFi/ Bluetooth coexistence"***, which is used to control the ESP32's Bluetooth and Wi-Fi coexistence using software, thus balancing the coexistence requirement for controlling the RF module by both the Wi-Fi and Bluetooth modules. Please note that if Option ***"Software controls WiFi/Bluetooth coexistence"*** is enabled, the BLE scan interval shall not exceed 0x100 slots (about 160 ms).

- If only the BLE and Wi-Fi coexistence is required, this option can be enabled or disabled. However, if this option is not enabled, please note that the "BLE scan interval - BLE scan window" should be larger than 150 ms, and the BLE scan interval should be less than 500 ms.
- If the Classic Bluetooth and Wi-Fi coexistence is required, it is recommended that you enable this option.

In ESP-IDF V3.0 and earlier versions, the performance of Classic Bluetooth and Wi-Fi coexistence might be poor at times. However, in ESP-IDF V3.1 and later versions, we has solved this issue. At the moment, ESP32 can simultaneously function as a Wi-Fi module and as a Bluetooth speaker, playing music smoothly.

### 5.3.3. How much memory does ESP32 Bluetooth use?

1. Controller:
   - BLE mode: 40 KB (.data + .bss + hardware)
   - BR/EDR mode: 65 KB (.data + .bss + hardware)
   - BT/BLE Dual-mode: 70 KB (.data + .bss + hardware)
2. Master device:
   - BLE:
     - GATT Client (Gatt Client demo): 24 KB (.bss+.data) + 23 KB (heap) = 47 KB
     - GATT Server (GATT Server demo): 23 KB (.bss+.data) + 23 KB (heap) = 46 KB
     - GATT Client & GATT Server: 24 KB (.bss+.data) + 24 KB (heap) = 48 KB
     - SMP: 5 KB

- Classic Bluetooth (Classic Bluetooth A2DP_SINK demo, with SMP/SDP/A2DP/AVRCP): 48 KB (.bss+.data) + 24 KB (heap) = 72 KB (with additional 13 KB for running the demo)

The above-mentioned Heap all contains the Task Stack because the Task Stack is allocated from the Heap and counts as a heap.

3. Optimized PSRAM version:

   In ESP-IDF V3.0 and later versions, if you configure the PSRAM related options in the menuconfig menu and then put some .bss/.data sections and heaps of Bluedroid (Host) into PSRAM, it can save additional 50 KB.

# 6. Wi-Fi and LwIP

## 6.1. Performance

### 6.1.1. How can ESP32 Wi-Fi performance be tested?

Please use the codes under the directory of example/wifi/iperf for testing.

### 6.1.2. What is ESP32 Wi-Fi throughput?

See *ESP32's Wi-Fi throughput*.

## 6.2. Memory

### 6.2.1. How much memory does Wi-Fi occupy after booting?

The memory used for Wi-Fi booting mainly includes the following components:

1. Wi-Fi tasks and queues: 6.144 KB
2. Wi-Fi AMPDU: 2.092 KB
3. Wi-Fi power management tasks and queues: 3.44 KB
4. Wi-Fi internal data structure: 1.348 KB
5. Event tasks and queues: 6.144 KB
6. LwIP tasks and mailboxes: 3.868 KB
7. The Wi-Fi/LwIP static RX/TX cache depends on the menuconfig configuration.
8. The Wi-Fi/LwIP dynamic RX/TX cache depends on the menuconfig configuration.

The memory occupied by Items 1~6 during the initialization totals to: 23.056 KB.

The memory occupied by items 7 and 8 changes dynamically and depends on the menuconfig configuration and the packet sending/receiving.

📖 ***Note:***

- *IDF V3.0 and later versions have optimized memory usage for item 2, saving 2.092 KB;*
- *IDF V3.1 and later versions have optimized memory usage for item 3, saving 3.44 KB.*

### 6.2.2. What sleep modes does ESP32 have? What is the difference between them?

ESP32 has three sleep modes: Modem-sleep, Light-sleep, and Deep-sleep.

- Modem-sleep:
  - The Station Legacy Fast sleep mode specified in the Wi-Fi specification, in which the Station sends a NULL frame to notify the AP to sleep or wake up.

    - After the station is connected to the AP, the station is automatically turned on. After the station enters the Modem-sleep mode, the RF module is shut down. During the modem sleep, the connection to the AP is maintained. After the AP disconnects from the station, Modem-sleep does not work.
    - After ESP32 enters Modem-sleep mode, the CPU clock frequency can be lowered to further reduce the current.
- Light-sleep:
    - A Station sleep mode based on Modem-sleep;
    - The differences between Light-sleep and Modem-sleep are:
        - After ESP32 enters the Light-sleep mode, not only the RF module but also the CPU and part of the system clock are suspended.
        - After ESP32 exits the Light-sleep mode, the CPU resumes working.
- Deep-sleep:
    - A sleep mode that is not specified in the Wi-Fi specification;
    - After ESP32 enters the Deep-sleep mode, all modules are closed except for RTC modules;
    - After ESP32 exits the Deep-sleep mode, the entire system reruns, which is similar to the system reboot;
    - During the deep sleep, no connection to the AP is maintained.

## 6.3. Scan

### 6.3.1. How to perform AP scanning when there is not enough memory?

The memory for storing the scan results is dynamically applied. If there is a large number of APs in the surrounding area, a lot of memory will be occupied. At this time, one possible solution is to scan channel by channel.

### 6.3.2. How much time does an ESP32 scan take?

The total time for scanning depends on:

- Active scan (by default) or passive scan.
- The time spent on each channel is 120 ms for active scanning and 360 ms for passive scanning.
- The country code and configured channel range from 1~13 channels (by default).
- Fast scan (by default) or full-channel scan.
- Station mode or Station-AP mode, and if any active connections are currently maintained.

By default, channels 1 to 11 use active scans, and channels 12 to 13 use passive scans.

- In the absence of connection in Station mode, the total time for a full-channel scan is: 11*120 + 2*360 = 2040 ms;
- With active connections in Station mode or Station-AP mode, the total time for a full-channel scan is: 11*120 + 2*360 + 13*30 = 2430 ms.

## 6.4. RX/TX

### 6.4.1. How can RAW 802.11 messages be sent?

The `esp_wifi_80211_tx` can be called to send messages (available in IDF v3.1).

## 6.5. LwIP

### 6.5.1. How soon can the associated resources be released after the TCP connection is closed?

The associated resources can be released in 20 seconds or can be specified by the sent `linger/send_timeout` parameter.

### 6.5.2. How many sockets can a LwIP create?

Up to 32 and 10 by default.

# 7. Digital

## 7.1. RTC

### 7.1.1. What should I do if there is a Brownout Reset?

1. Check if the power supply is faulty, for example, the voltage drops below the threshold.
2. Since the current fluctuations during the chip operation is relatively large, please make sure that the output of the power supply is sufficient. If user uses an USB-cable to power the ESP board, the USB-cable should be high-quality and capable of delivering the required current. For details, please refer to *ESP32 Datasheet*.

## 7.2. Bluetooth

### 7.2.1. What is the transmit power of ESP32 Bluetooth?

ESP32 Bluetooth has 9 transmit power levels, corresponding to -12 ~ 12 dBm, with a 3 dBm interval. The controller software limits the transmit power and selects the power level according to the corresponding power level declared by the product.

# 8. Peripherals

## 8.1. Pin

### 8.1.1. Which pins should be paid attention when being used?

IO6 ~ IO11 are used for connecting flash and cannot be used as GPIOs.

GPIO16 and GPIO17 of a WROVER module will be occupied by the system and cannot be used as GPIOs.

Also, ESP32 has five strapping pins, which should also be paid attention to when being used. For details, please refer to *ESP32 Datasheet*.

## 8.2. Touch Sensor

### 8.2.1. The data reading is very unstable when ESP32 is used in touch-related applications. Where can I find more information for reference designs?

Please refer to the recommended hardware and software reference designs at: *https://github.com/espressif/esp-iot-solution/tree/master/examples/touch_pad_evb*.

## 8.3. Debug

### 8.3.1. How can I make sending messages by UART0 disabled by default?

Connecting GPIO15 to Ground helps with the bootloader messages. And to also suppress the ESP-IDF messages, users need to select the appropriate log option in menuconfig / Component config /Log output.