ESP8266 Reset Causes and Common Fatal Exception Causes

Version 1.0

About This Guide

This guide introduces the methods of identifying the causes of ESP8266 reset and common Fatal exceptions.

The guide structure is as follows:

Chapter	Title	Content
Chapter 1	Reset Causes	Introduction to two methods of identifying ESP8266 reset causes: ROM code and user program.
Chapter 2	Common Fatal Exceptions and Causes	Description of common ESP8266 Fatal exceptions and their causes.

Release Notes

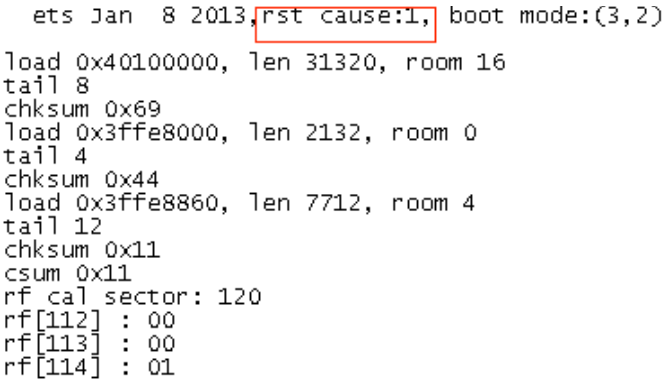
Date	Version	Release notes
2016.08	V1.0	Initial Release.

Table of Contents

1. Reset Causes

1.1. Identifying Reset Cause in ROM Code

Each time ESP8266 reboots, the ROM code will print out a number corresponding to the reset cause, as the following figure shows. You can verify the cause of the reset based on the number. Use this as a debugging method when you cannot start the user program and need to analyze the cause of the reset.



The following table shows reset causes printed in ROM code.

Table 1-1. Identifying Reset Cause in ROM Code

Rst cause No.	Cause
0	Undefined
1	Power reboot
2	External reset or wake-up from Deep-sleep
4	Hardware WDT reset

> ⚠ ***Notice:***
>
> *The reboot state will not change after software WDT reset or software reset. For example, when the first reset is caused by a power reboot, the rst cause number is 1. After software reset, the rst cause number will still be 1.*

1.2. Identifying Reset Cause Using User Program

You can also identify the reset cause by adding an application layer program, which provides relatively comprehensive analysis of the reset cause. Use this method when garbled output is printed where crash occurs and can not be debugged.

Add the following code segment:

```
struct rst_info *rtc_info = system_get_rst_info();
```

```
os_printf("reset reason: %x\n", rtc_info->reason);

        if (rtc_info->reason == REASON_WDT_RST ||

                rtc_info->reason == REASON_EXCEPTION_RST ||

                rtc_info->reason == REASON_SOFT_WDT_RST) {

                if (rtc_info->reason == REASON_EXCEPTION_RST) {

                        os_printf("Fatal exception (%d):\n", rtc_info->exccause);

                }

                os_printf("epc1=0x%08x, epc2=0x%08x, epc3=0x%08x, excvaddr=0x%08x,
depc=0x%08x\n",

                                rtc_info->epc1, rtc_info->epc2, rtc_info->epc3, rtc_info-
>excvaddr, rtc_info->depc);//The address of the last crash is printed, which is used to
debug garbled output.

        }
```

For information on `system_get_rst_info()` and associated data structures, please refer to *ESP8266 Non-OS SDK API Reference* and *ESP8266 RTOS SDK API Reference* (link: *espressif.com/en/support/download/documents*).

The following table shows the reset causes identified by adding user program.

Table 1-2. Identifying Reset Cause Using User Program

Rst cause No.	Cause	GPIO state
0	Power reboot	Changed
1	Hardware WDT reset	Changed
2	Fatal exception	Unchanged
3	Software watchdog reset	Unchanged
4	Software reset	Unchanged
5	Deep-sleep	Changed
6	Hardware reset	Changed

2. Common Fatal Exceptions and Causes

When a program crashes, you can debug the crash based on the Fatal exception number. The following table shows common Fatal exceptions and their possible causes.

Table 2-1. Common Fatal Exceptions and Causes

Fatal exception No.	Description	Possible Causes
0	Invalid command	1. Damaged BIN binaries 2. Wild pointers
6	Division by zero	Division by zero
9	Unaligned read/write operation addresses	1. Unaligned read/write Cache addresses 2. Wild pointers
28/29	Access to invalid address	1. Access to Cache after it is turned off 2. Wild pointers

For example:

```
Fatal exception (28):
epc1=0x4025bfa6, epc2=0x00000000, epc3=0x00000000, excvaddr=0x0000000f,
depc=0x00000000
```

- If ***user1.1024.new.2.bin*** is used, verify the exception address "0x4025bfa6" in the ***user1.1024.new.2.S*** file. Add print to the user's code to debug the Fatal exception.
- If ***eagle.irom0text.bin*** is used, verify the cause of the Fatal exception in the ***eagle.S*** file.
- If the address of exception cannot be found, it means that the crash occurs during an interrupt, or that there is a code problem in ROM, such as:
  - ***4000e190 <memset>***
  - ***4000df48 <memcpy>***
  - ***4000dea8 <memcmp>***
  - ***4000de84 <bzero>***
  - ***4000e1e0 <strstr>***

Espressif IOT Team

www.espressif.com