ESP32

Series SoC Errata Version 2.8

ESPRESSIF

Table of contents

1 Chip Revision Identification

Espressif is introducing a new **vM.X** numbering scheme to indicate chip revisions. This guide outlines the structure of this scheme and provides information on chip errata and additional identification methods.

1.1 Chip Revision Numbering Scheme

The new numbering scheme **vM.X** consists of the major and minor numbers described below.

M –Major number, indicating the major revision of the chip product. If this number changes, it means the software used for the previous version of the product is incompatible with the new product, and the software version shall be upgraded for the use of the new product.

X –Minor number, indicating the minor revision of the chip product. If this number changes, it means the software used for the previous version of the product is compatible with the new product, and there is no need to upgrade the software.

The **vM.X** scheme replaces previously used chip revision schemes, including ECOx numbers, Vxxx, and other formats if any.

1.2 Primary Identification Methods

eFuse Bits

The chip revision is encoded using four eFuse fields:

- EFUSE_BLK0_RDATA5[25:24]
- EFUSE_BLK0_RDATA5[20]
- EFUSE_BLK0_RDATA3[15]
- APB_CTRL_DATE[31]

Table 1.1: Chip Revision Identification by eFuse Bits

	eFuse Bit	Chip Revision				
		v0.0	v1.0	v1.1	v3.0	v3.1
Major Number	APB_CTRL_DATE[31]	0	0	0	1	1
	EFUSE_BLK0_RDATA5[20]	0	0	0	1	1
	EFUSE_BLK0_RDATA3[15]	0	1	1	1	1
Minor Number	EFUSE_BLK0_RDATA5[25]	0	0	0	0	0
	EFUSE_BLK0_RDATA5[24]	0	0	1	0	1

Chip Marking

- **Espressif Tracking Information** line in chip marking

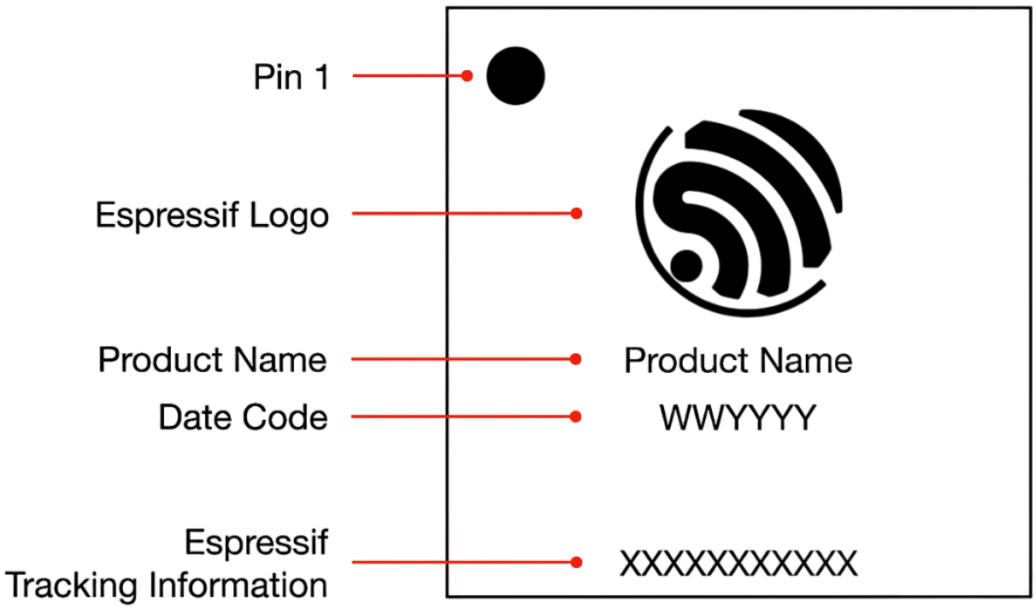

Figure 1.1: Chip Marking Diagram

Table 1.2: Chip Revision Identification by Chip Marking

Chip Revision	Espressif Tracking Information
v0.0	XXXXXXXX
v1.0	X B XXXXXX
v1.1	X F XXXXXX
v3.0	X E XXXXXX
v3.1	X G XXXXXX

Module Marking

- **Specification Identifier** line in module marking

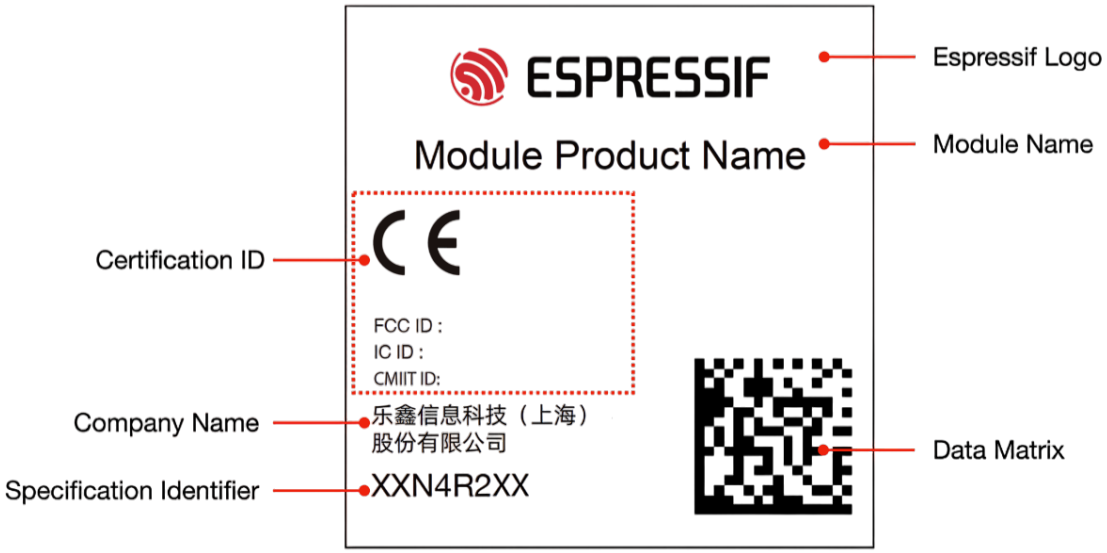

Figure 1.2: Module Marking Diagram

Table 1.3: Chip Revision Identification by Module Marking

Chip Revision	Specification Identifier
v0.0	XXXXXX [1]
v1.0	XXXXXX
v1.1	MF XXXX
v3.0	ME XXXX
v3.1	MG XXXX

[1] To distinguish between v0.0 and v0.1, please check the eFuse bits.

1.3 Additional Identification Methods

Date Code

Some errors in the chip product don't need to be fixed at the silicon level, or in other words in a new chip revision.

In this case, the chip may be identified by **Date Code** in chip marking (see *Chip Marking Diagram*). For more information, please refer to Espressif Chip Packaging Information.

PW Number

Modules built around the chip may be identified by **PW Number** in product label (see *Module Product Label*). For more information, please refer to Espressif Module Packaging Information.

Figure 1.3: Module Product Label

Note: Please note that **PW Number** is only provided for reels packaged in aluminum moisture barrier bags (MBB).

1.4 ESP-IDF Release Compatibility

Information about ESP-IDF release that supports a specific chip revision is provided in Compatibility Between ESP-IDF Releases and Revisions of Espressif SoCs.

1.5 Related Documents

- For more information about the chip revision upgrade and their identification of series products, please refer to ESP32 Product/Process Change Notifications (PCN).
- For more information about the chip revision numbering scheme, see Compatibility Advisory for Chip Revision Numbering Scheme.

2 Errata Summary

Table 2.1: Errata summary

Category	Descriptions	Affected Revisions				
		v0.0	v1.0	v1.1	v3.0	v3.1
CPU	*[CPU] The CPU crashes when the clock frequency switches*	Y				
	[CPU] CPU has limitations when accessing peripherals in chips	Y	Y	Y	Y	Y
	[CPU] There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces	Y	Y	Y	Y	Y
	[CPU] When the CPU accesses external SRAM in a certain sequence, read and write errors may occur		Y	Y		
	[CPU] When the CPU accesses external SRAM through cache, under certain conditions read and write errors occur	Y				
	[CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur	Y	Y	Y		
	[CPU] When a CPU is interrupted while accessing five specific FIFO registers, subsequent CPU accesses will get halted	Y	Y	Y	Y	Y
	[CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost	Y				
ULP	*[ULP] ULP coprocessor and touch sensors can not be used during Deep-sleep when RTC_PERIPH power domain is up*	Y	Y	Y	Y	Y
GPIO	*[GPIO] For pads with both GPIO and RTC_GPIO functionality, the GPIO pull-up and pull-down configuration register fields are nonfunctional*	Y	Y	Y	Y	Y
	[GPIO] Within the same group of GPIO pins, edge interrupts cannot be used together with other interrupts	Y	Y	Y	Y	Y
	[GPIO] When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns	Y	Y	Y	Y	Y
Reset	*[Reset] The Brown-out Reset (BOR) function does not work*	Y				
	[Reset] A spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep	Y				
	[Reset] Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deepsleep	Y	Y	Y		

Table 2.2: Errata summary

Category	Descriptions	Affected Revisions				
		v0.0	v1.0	v1.1	v3.0	v3.1
Clock	*[Clock] Audio PLL frequency range is limited*	Y				
	[Clock] ESP32 cannot be used as the PHY clock source if Wi-Fi and Ethernet are used at the same time	Y	Y	Y	Y	Y
RTC	*[RTC] RTC Register Read Error After Wake-up from Light-sleep Mode*	Y	Y	Y	Y	Y
Watchdog	*[Watchdog] ESP32 chip may have a live lock under certain conditions that will cause interrupt watchdog issue*				Y	Y
UART	*[UART] UART fifo_cnt does not indicate the data length in FIFO correctly*	Y	Y	Y	Y	Y
TWAI	*[TWAI] After losing arbitration, a dominant bit on the 3rd bit of intermission is not interpreted as an SOF*	Y	Y	Y	Y	Y
	[TWAI] Message transmitted after bus-off recovery is erroneous	Y	Y	Y	Y	Y
	[TWAI] When the 8th bit of the error delimiter is dominant, the error passive state is not entered	Y	Y	Y	Y	Y
	[TWAI] Error status bit is not frozen during bus-off recovery	Y	Y	Y	Y	Y
	[TWAI] Receiving an erroneous data frame can cause the data bytes of the next received data frame to be invalid	Y	Y	Y	Y	Y
	[TWAI] A negative phase error where |e| > SJW(N) will cause the remaining transmitted bits to be left shifted	Y	Y	Y	Y	Y
	[TWAI] Receive Error Counter (REC) is allowed to change whilst in reset mode or bus-off recovery	Y	Y	Y	Y	Y
	[TWAI] When the RX FIFO overruns with 64 or more messages, the RX FIFO becomes unrecoverable	Y	Y	Y	Y	Y
	[TWAI] Suspend transmission is included even after losing arbitration	Y	Y	Y	Y	Y
	[TWAI] When a stuff error occurs during arbitration whilst being transmitter, any errors in the subsequent error/overload frame will not increase the TEC	Y	Y	Y	Y	Y
	[TWAI] Reading the interrupt register may lead to a transmit interrupt being lost	Y	Y	Y	Y	Y
LEDC	*[LEDC] When the LEDC is in decremental fade mode, a duty overflow error may occur*	Y	Y	Y	Y	Y

3 All Errata Descriptions

3.1 [CPU] The CPU crashes when the clock frequency switches

Affected revisions: v0.0

Description

The CPU crashes when the clock frequency switches directly from 240 MHz to 80/160 MHz

Workarounds

When switching frequencies, use intermediate frequencies as follows:

1. 2 MHz <-> 40 MHz <-> 80 MHz <-> 160 MHz
2. 2 MHz <->40 MHz <->240 MHz

Solution

Fixed in chip revision v1.0.

3.2 [CPU] CPU has limitations when accessing peripherals in chips

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

As described in *[CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost*, *[CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur*, *[CPU] There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces*, CPU has limitations when accessing peripherals in chips of different revisions using 0x3FF0_0000 ~ 0x3FF1_EFFF, 0x3FF4_0000 ~ 0x3FF7_FFFF, and 0x6000_0000 ~ 0x6003_FFFF.

<table>
<tr><th rowspan="2">Address space (Bus)</th><th rowspan="2">Register type</th><th rowspan="2">Operation</th><th colspan="4">Chip Revision</th></tr>
<tr><th>v0.0</th><th>v1.0</th><th>v1.1</th><th>v3.0 v3.1</th></tr>
<tr><td rowspan="4">0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF (DPORT)</td><td rowspan="2">Non-FIFO</td><td>Write</td><td colspan="3">Yes</td><td>Yes</td></tr>
<tr><td>Read</td><td colspan="3">No (refer to [CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur)</td><td>Yes</td></tr>
<tr><td rowspan="2">FIFO</td><td>Write</td><td>No (refer to [CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost)</td><td colspan="3">Yes</td></tr>
<tr><td>Read</td><td>Yes</td><td colspan="3">Yes</td></tr>
<tr><td rowspan="4">0x6000_0000 ~ 0x6003_FFFF (AHB)</td><td rowspan="2">Non-FIFO</td><td>Write</td><td colspan="4">Yes</td></tr>
<tr><td>Read</td><td colspan="4">Yes</td></tr>
<tr><td rowspan="2">FIFO</td><td>Write</td><td colspan="4">Yes</td></tr>
<tr><td>Read</td><td colspan="4">No (No such feature, unpredictable results)</td></tr>
</table>

Note:

- Yes: operation is executed correctly
- No: operation fails

3.3 [CPU] There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

1. The CPU read operations that fall in these two address spaces are speculative. Speculative read operations can cause the behavior described by the program to be inconsistent with the actual behavior of the hardware.
2. If the two CPUs continuously access address space 0x3FF0_0000 ~ 0x3FF1_EFFF at the same time, some of the access may be lost.
3. When the CPU reads FIFO through the address space 0x3FF4_0000 ~ 0x3FF7_0000, the FIFO read pointer is updated with delays. As the CPU frequency increases, the interval between two consecutive FIFO reads initiated by the CPU is shortened. When a new FIFO read request arrives, the FIFO read pointer has not been updated, causing the CPU to read the value of the previous FIFO read operation.

Workarounds

1. Insert "MEMW" instruction before the CPU access operation that falls in these two address spaces. That is, in C/C++, software needs to always use the "volatile" attribute when accessing registers in these two address spaces.
2. When the CPU frequency is 160 MHz, add six "nop" between two consecutive FIFO reads. When the CPU frequency is 240 MHz, add seven "nop" between two consecutive FIFO reads.

Solution

No fix scheduled.

3.4 [CPU] When the CPU accesses external SRAM in a certain sequence, read and write errors may occur

Affected revisions: v1.0 v1.1

Description

This error may occur when the CPU executes the following instructions to access external SRAM:

```
store.x at0, as0, n
load.y at1, as1, m
```

In the pseudo-assembly instructions above, `store.x` represents an x-bit write operation, while `load.y` represents a y-bit read operation. `as0+n` and `as1+m` represent the same address in external SRAM.

- The instructions can be sequential or contained within the same pipeline (less than four intermediate instructions, and no pipeline flushes.)
- When x>=y, the data write may be lost. (NOTE: when both the `load` and the `store` refer to 32-bit values, the write is only lost if an interrupt occurs between the first and second instructions.)
- When x<y, data writes may be lost and invalid data may be read.

Workarounds

This bug is automatically worked around when external SRAM use is enabled in ESP-IDF v3.0 and newer.

- When x>=y, insert four nop instructions between `store.x` and `load.y`.
- When x<y, insert a memw instruction between `store.x` and `load.y`.

Solution

Fixed in chip revision v3.0.

3.5 [CPU] When the CPU accesses external SRAM through cache, under certain conditions read and write errors occur

Affected revisions: v0.0

Description

Access to external SRAM through cache will cause read and write errors if these operations are pipelined together by the CPU.

Workarounds

If accessing external SRAM from chip revision v0.0, users must ensure that access is always one-way—only a write or a read can be in progress at a single time in the CPU pipeline.

The MEMW instruction can be used: insert __asm__(“MEMW”) after any read from external PSRAM that may be followed by a write to PSRAM before the CPU pipeline is flushed.

Solution

Fixed in chip revision v1.0.

3.6 [CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur

Affected revisions: v0.0 v1.0 v1.1

Description

Running in dual-core CPU mode, when one CPU bus reads address space A (0x3FF0_0000 ~ 0x3FF1_EFFF), while the other CPU bus reads address space B (0x3FF4_0000 ~ 0x3FF7_FFFF), an incorrect read may be generated on the CPU reading address space B.

Workarounds

Either of the following workarounds can be used:

- When either CPU reads address space A, prevent the other CPU bus from reading address space B via locks and interrupts.
- Before reading address space A, disable interrupts and insert a read from address space B on the same CPU (read a non-FIFO register, e.g., 0x3FF40078).

Solution

Fixed in chip revision v3.0.

3.7 [CPU] When a CPU is interrupted while accessing five specific FIFO registers, subsequent CPU accesses will get halted

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the CPU attempts to read five FIFO registers 0x3FF40000 (UART0), 0x3FF50000 (UART1), 0x3FF6E000 (UART2), 0x3FF4F004 (I2S0), and 0x3FF6D004 (I2S1), and if an interrupt occurs, the read request will be interrupted. This will cause the bus bridge to be stuck in a state of waiting for the read request to end. Consequently, the subsequent access to the APB peripheral registers (0x3FF40000 ~ 0x3FF7FFFF or 0x60000000 ~ 0x6003FFFF) by any CPUs will be rejected and halted.

Writing to these five FIFO registers does not have such an issue.

Workarounds

Disable CPU interrupts before reading these five FIFO registers. Enable CPU interrupts after read access.

Solution

No fix scheduled.

3.8 [CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost

Affected revisions: v0.0

Description

Some ESP32 peripherals are mapped to two internal memory buses (AHB & DPORT). When written via DPORT, consecutive writes to the same address may be lost.

Workarounds

When writing the same register address (i.e., FIFO-like addresses) in sequential instructions, use the equivalent AHB address not the DPORT address. (For other kinds of register writes, using DPORT registers will give better write performance.)

Registers	DPORT Addresses	AHB (Safe) Addresses
UART_FIFO_REG	0x3FF40000	0x60000000
UART1_FIFO_REG	0x3FF50000	0x60010000
UART2_FIFO_REG	0x3FF6E000	0x6002E000
I2S0_FIFO_RD_REG	0x3FF4F004	0x6000F004
I2S1_FIFO_RD_REG	0x3FF6D004	0x6002D004
GPIO_OUT_REG	0x3FF44004	0x60004004
GPIO_OUT_W1TS_REG	0x3FF44008	0x60004008
GPIO_OUT_W1TC_REG	0x3FF4400C	0x6000400C
GPIO_OUT1_REG	0x3FF44010	0x60004010
GPIO_OUT1_W1TS_REG	0x3FF44014	0x60004014
GPIO_OUT1_W1TC_REG	0x3FF44018	0x60004018
GPIO_ENABLE_REG	0x3FF44020	0x60004020
GPIO_ENABLE_W1TS_REG	0x3FF44024	0x60004024
GPIO_ENABLE_W1TC_REG	0x3FF44028	0x60004028
GPIO_ENABLE1_REG	0x3FF4402C	0x6000402C
GPIO_ENABLE1_W1TS_REG	0x3FF44030	0x60004030
GPIO_ENABLE1_W1TC_REG	0x3FF44034	0x60004034

Solution

Fixed in chip revision v1.0.

Note: Software cannot use AHB addresses to read FIFO.

3.9 [ULP] ULP coprocessor and touch sensors can not be used during Deep-sleep when RTC_PERIPH power domain is up

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

FAST_CLK is the main working clock for ULP coprocessor and touch sensors. But if RTC_PERIH power domain is powered on, the ULP coprocessor and touch sensors will receive the startup signal earlier than the clock management module, which causes the ULP coprocessor and touch sensors working under SLOW_CLK for a period of time, resulting in inaccurate working clock.

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.10 [GPIO] For pads with both GPIO and RTC_GPIO functionality, the GPIO pull-up and pull-down configuration register fields are nonfunctional

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

GPIO pull-up and pull-down resistors for pads with both GPIO and RTC_GPIO functionality can only be controlled via RTC_GPIO registers.

Workarounds

Use RTC_GPIO registers for both GPIO and RTC_GPIO functions.

Solution

This issue is automatically worked around when using GPIO drivers in ESP-IDF v2.1 or newer.

3.11 [GPIO] Within the same group of GPIO pins, edge interrupts cannot be used together with other interrupts

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

GPIO0 ~ GPIO31 share a set of interrupt configuration registers and belong to one group, GPIO32 ~ GPIO39 share another set of registers and belong to another group, and RTC GPIO0 ~ GPIO17 share yet another set of registers and belong to yet another group. If one GPIO pad within a group is configured with edge-triggered interrupt, then other interrupts (including both edge-triggered and level-triggered interrupts within that group cannot be configured.

There is no such limitation for level-triggered interrupts, which means, if there are no edgetriggered interrupts configured within a group, then there can be any number of leveltriggered interrupts in that group.

Reason

When the following three sets of STATUS/W1TS/W1TC registers for GPIOs are being operated, edge-triggered interrupts may not be properly triggered within the same group.

- When the following registers are being operated, edge-triggered interrupts for GPIO_STATUS_REG may not be properly triggered:
  - GPIO_STATUS_W1TS_REG
  - GPIO_STATUS_W1TC_REG
  - GPIO_STATUS_REG
- When the following registers are being operated, edge-triggered interrupts for GPIO_STATUS1_REG may not be properly triggered:
  - GPIO_STATUS1_W1TS_REG
  - GPIO_STATUS1_W1TC_REG
  - GPIO_STATUS1_REG
- When the following registers are being operated, edge-triggered interrupts for RTCIO_RTC_GPIO_STATUS_REG may not be properly triggered:
  - RTCIO_RTC_GPIO_STATUS_W1TS_REG
  - RTCIO_RTC_GPIO_STATUS_W1TC_REG
  - RTCIO_RTC_GPIO_STATUS_REG

Workarounds

Simulate edge-triggered interrupts using level-triggered interrupts, as outlined below.

To trigger a GPIO interrupt on a rising edge, follow the steps:

1. Set the GPIO interrupt type to high.
2. After the CPU services the interrupt, change the GPIO interrupt type to low. A second interrupt occurs at this time, and the CPU needs to ignore the interrupt service routine.

To trigger a GPIO interrupt on a falling edge, follow the steps:

1. Set the GPIO interrupt type to low.
2. After the CPU services the interrupt, change the GPIO interrupt type to high. A second interrupt occurs at this time, and the CPU needs to ignore the interrupt service routine.

Solution

No fix scheduled.

3.12 [GPIO] When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

Powering on the following RTC peripherals will trigger this issue:

- SAR ADC1
- SAR ADC2
- AMP

Workarounds

When enabling power for any of these peripherals, ignore input from GPIO36 and GPIO39.

Solution

No fix scheduled.

3.13 [Reset] The Brown-out Reset (BOR) function does not work

Affected revisions: v0.0

Description

The Brown-out Reset (BOR) function does not work. The system fails to boot up after BOR.

Workarounds

There is no workaround for this issue.

Solution

Fixed in chip revision v1.0.

3.14 [Reset] A spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep

Affected revisions: v0.0

Description

A spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep.

Workarounds

To work around the watchdog reset when waking from Deep-sleep, the CPU can execute a program from RTC fast memory. This program must clear the illegal access flag in the cache MMU as follows:

1. Set the PRO_CACHE_MMU_IA_CLR bit in DPORT_PRO_CACHE_CTRL1_REG to 1.
2. Clear this bit.

During initial power-up the spurious watchdog reset cannot be worked around, but ESP32 will boot normally after this reset.

Solution

Fixed in chip revision v1.0.

3.15 [Reset] Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deepsleep

Affected revisions: v0.0 v1.0 v1.1

Description

If the ESP32 reads from the flash chip before it is ready, invalid data can cause booting to fail until a Watchdog Timer reset occurs. This can occur on power-on and on wake from Deep-sleep, if the ESP32 VDD_SDIO is used to power the flash chip.

Workarounds

1. Replace the flash chip with one with a fast start-up time (<800 µs from power-on to ready to read). This works around the issue for both power-on and wake from Deepsleep.
2. When waking from Deep-sleep, this issue is automatically worked around in ESP-IDF v2.0 and newer (the delay to wait can be configured if necessary). In this workaround, the CPU executes from RTC fast memory immediately after waking and a delay is added before continuing to read the program from flash.

Solution

Fixed in chip revision v3.0.

3.16 [Clock] Audio PLL frequency range is limited

Affected revisions: v0.0

Description

When configuring the Audio PLL, configuration registers sdm0 & sdm1 are not used. This limits the range and precision of PLL frequencies which can be configured.

For chip revision v0.0, the Audio PLL frequency is calculated in hardware as follows:

$$f_{out} = \frac{f_{xtal} \times (sdm2 + 4)}{2 \times (odiv + 2)}$$

For chip revision v1.0 onwards this bug is fixed and the Audio PLL frequency is calculated in hardware as follows:

$$f_{out} = \frac{f_{xtal} \left(sdm2 + \frac{sdm1}{2^8} + \frac{sdm0}{2^{16}} + 4\right)}{2 \times (odiv + 2)}$$

Workarounds

The particular hardware frequency calculation is automatically accounted for when setting Audio PLL frequency via the I2S driver in ESP-IDF v3.0 and newer. However, the range and precision of available Audio PLL frequencies is still limited when using chip revision v0.0.

Solution

Fixed in chip revision v1.0.

3.17 [Clock] ESP32 cannot be used as the PHY clock source if Wi-Fi and Ethernet are used at the same time

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

In RMII working mode, the Ethernet MAC and PHY require a common 50 MHz reference clock (i.e., the RMII clock). If Wi-Fi and Ethernet are used simultaneously, the RMII clock cannot be generated by the internal APLL clock, as it would result in clock instability.

Workarounds

1. If you want to use the internal APLL to generate the reference clock, you need to disable Wi-Fi.
2. If you want to use both Ethernet and Wi-Fi simultaneously, you need to use an external PHY or external clock source to provide the reference clock.

Solution

No fix scheduled.

3.18 [RTC] RTC Register Read Error After Wake-up from Light-sleep Mode

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

If an RTC peripheral is turned off in Light-sleep mode, there is a certain probability that after waking up from Light-sleep, the CPU of ESP32 will read the registers in the RTC power domain incorrectly.

Workarounds

Users are suggested not to power down RTC peripherals in Light-sleep mode. There will be no impact on power consumption.

Solution

No fix scheduled.

3.19 [Watchdog] ESP32 chip may have a live lock under certain conditions that will cause interrupt watchdog issue

Affected revisions: v3.0 v3.1

Description

On ESP32 chip revision v3.0, when the following conditions are met at the same time, a live lock will occur, causing the CPUs to get stuck in the state of memory access and stop executing instructions.

1. Dual-core system.
2. Of the four Instruction/Data buses (IBUS/DBUS) that access external memory, three simultaneously initiate access requests to the same cache set, and all three requests result in cache misses.

Workarounds

When a live lock occurs, software proactively or passively recognizes and unlocks the cache line contention, and then the two cores complete their respective cache operations one after another, following a first-come, first-served policy, to resolve the live lock. The detailed process is as follows:

1. If the live lock occurs when the instructions executed by the two cores are not in the critical section of the code, the various types of system interruptions will proactively release the cache line competition and resolve the live lock.
2. If the live lock occurs when the instructions executed by the two cores are located in the critical section of the code, the system will mask interrupts at level 3 and below. Therefore, software needs to set up a high priority (level 4 or 5) interrupt for each core in advance, connect the interrupts to the same timer, and configure an appropriate timeout threshold. The timer timeout interrupt generated by the live lock will force both cores to enter the high-priority interrupt handler, thereby releasing the IBUS of both cores to resolve the live lock. The live lock resolution process is completed in three stages:
    a. In the first stage, both cores wait for the CPU write buffer to be cleared.
    b. In the second stage, one core (Core 0) waits and the other core (Core 1) executes instructions.
    c. In the third stage, Core 1 waits and Core 0 executes instructions.

Solution

No fix scheduled.

3.20 [UART] UART fifo_cnt does not indicate the data length in FIFO correctly

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When software uses DPORT to read UART fifo_cnt, and such operation is interrupted, then fifo_cnt will decrement by 1 erroneously.

Workarounds

When using DPort to read fifo, calculate the real count based on the FIFO read and write offset address. For example:

```
if (wr_addr > rd_addr) {
    len = wr_addr - rd_addr;
} else if (wr_addr < rd_addr){
    len = (wr_addr + 128) - rd_addr;
} else {
    len = fifo_cnt > 0 ? 128 : 0;
}
```

In the above code snippet, `wr_addr` represents the FIFO write offset address, `rd_addr` represents the FIFO read offset address, `fifo_cnt` represents the number of valid bytes in the FIFO, `len` represents the correct number of valid bytes after calculation.

Solution

No fix scheduled.

3.21 [TWAI] After losing arbitration, a dominant bit on the 3rd bit of intermission is not interpreted as an SOF

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

The CAN2.0B protocol stipulates that a dominant bit on the 3rd bit of intermission shall be interpreted as a Start of Frame (SOF). Therefore, nodes shall begin receiving or transmitting (i.e., competing for arbitration) the ID field on the next bit.

When the TWAI controller loses arbitration and the following intermission's 3rd bit is dominant, the TWAI controller will not interpret this as an SOF and will make no attempt to compete for arbitration (i.e., does not retransmit its frame).

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.22 [TWAI] Message transmitted after bus-off recovery is erroneous

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

Upon completion of bus-off recovery, the next message that the TWAI controller transmits may be erroneous (i.e., does not adhere to TWAI frame format).

Workarounds

Upon detecting the completion of bus-off recovery (via the error warning interrupt), the TWAI controller should enter then exit reset mode so that the controller's internal signals are reset.

Solution

No fix scheduled.

3.23 [TWAI] When the 8th bit of the error delimiter is dominant, the error passive state is not entered

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the TWAI controller is the transmitter and has a TEC value between 120 and 127, transmitting an error frame will increment its TEC by 8 thus make the controller error passive (due to TEC becoming >= 128). However, if the 8th bit of the error delimiter is dominant, the TEC will still increment by 8 but the controller will not become error passive. Instead, the controller will become error passive when another error frame is transmitted. Note that the controller will still generate the required overload frame due to the dominant 8th bit.

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.24 [TWAI] Error status bit is not frozen during bus-off recovery

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the TWAI controller undergoes the bus-off recovery process, the controller must monitor 128 occurrences of the bus free signal (11 consecutive recessive bits) before it can become error active again. The number of bus-free signals remaining is indicated by the transmit error counter (TEC). Because the error status bit is not frozen during bus-off recovery, its value will change when the transmit error counter drops below the user-defined transmit error warning limit (96 by default) thus trigger the error warning limit interrupt before bus-off recovery has completed.

Workarounds

When undergoing bus-off recovery, an error warning interrupt does not necessarily indicate the completion of recovery. Users should check the STATUS_NODE_BUS_OFF bit to verify whether bus-off recovery has completed.

Solution

No fix scheduled.

3.25 [TWAI] Receiving an erroneous data frame can cause the data bytes of the next received data frame to be invalid

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the TWAI controller is receiving a data frame and a bit or stuff error occurs in the data or CRC fields, some data bytes of the next received data frame may be shifted or lost. Therefore, the next received data frame (including those filtered out by the acceptance filter) should be considered invalid.

Workarounds

Users can detect the errata triggering condition (i.e., bit or stuff error in the data or CRC field) by setting the INTERRUPT_BUS_ERR_INT_ENA and checking the ERROR_CODE_CAPTURE_REG when a bus error interrupt occurs. If the errata condition is met, the following workarounds are possible:

- The TWAI controller can transmit a dummy frame with 0 data bytes to reset the controller's internal signals. It is advisable to select an ID for the dummy frame that can be filtered out by all nodes on the TWAI bus.
- Hardware reset the TWAI controller (will require saving and restoring the current register values).

Solution

No fix scheduled.

3.26 [TWAI] A negative phase error where |e| > SJW(N) will cause the remaining transmitted bits to be left shifted

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the TWAI controller encounters a recessive to dominant edge with a negative phase error (i.e., the edge is early), it will correct for the phase error using resynchronization as required by the CAN2.0B protocol. However, if the TWAI controller is acting as transmitter and encounters a negative phase error where $e < 0$ and $|e| > SJW$, the bits transmitted following the phase error will be left shifted by one bit. Thus, the transmitted frame's contents (i.e., DLC, data bytes, CRC sequence) will be corrupted.

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.27 [TWAI] Receive Error Counter (REC) is allowed to change whilst in reset mode or bus-off recovery

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the TWAI controller enters reset mode (e.g., by setting the RESET_MODE bit or due to a bus-off condition) or when the TWAI controller undergoes bus-off recovery, the REC is still permitted to change. This can lead to the following cases:

- Whilst in reset mode or bus-off recovery, a changing REC can lead to the error status bit changing which in turn could trigger the error warning limit interrupt.
- During bus-off recovery, an REC > 0 can prevent the bus-off recovery process from completing.

Workarounds

When entering reset mode, the TWAI controller should set the LISTEN_ONLY_MODE to freeze the REC. The desired mode of operation should be restored before exiting reset mode or when bus-off recovery completes.

Solution

No fix scheduled.

3.28 [TWAI] When the RX FIFO overruns with 64 or more messages, the RX FIFO becomes unrecoverable

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When the RX FIFO overruns with multiple messages, and the RX message counter reaches 64, the RX FIFO will become unrecoverable. Any message read from the RX FIFO will be invalid. Attempting to release a message from the RX FIFO will have no effect.

Workarounds

The TWAI controller must be reset by software in order to recover the RX FIFO.

Solution

This issue is automatically worked around in ESP-IDF v4.3 and newer.

3.29 [TWAI] Suspend transmission is included even after losing arbitration

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

The CAN2.0B protocol stipulates that an error passive node that was the transmitter of a message shall add a suspend transmission field within the subsequent interframe space. However, error passive receivers shall not add a suspend transmission field.

When the TWAI controller is error passive and loses arbitration (hence becomes a receiver), it will still add a suspend transmission field in the subsequent interframe space. This results in the TWAI controller being late to start retransmission. Therefore, if another node transmits immediately after the interframe space is over, the TWAI controller will fail to compete for arbitration due to the other nodes not including a suspend transmission field in their interframe space (as per CAN2.0B specification).

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.30 [TWAI] When a stuff error occurs during arbitration whilst being transmitter, any errors in the subsequent error/overload frame will not increase the TEC

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

When a stuff error occurs during arbitration whilst being transmitter, the CAN2.0B protocol stipulates that an error frame be transmitted but the TEC should not increase (Exception 2 of Rule 3). The TWAI controller is able to fulfill these requirements without issue.

However, errors within the subsequent error/overload frames themselves will fail to increase the TWAI controller's TEC. Therefore, when a stuff error occurs during arbitration whilst being transmitter, the TEC will fail to increase in the following cases:

- Bit error in an active error flag or overload flag (Rule 4).
- Detecting too many dominant bits after the transmission of active error, passive error flag, and overload flags (Rule 6).

Workarounds

There is no workaround for this issue.

Solution

No fix scheduled.

3.31 [TWAI] Reading the interrupt register may lead to a transmit interrupt being lost

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

The TWAI controller's interrupt signals are cleared by reading the INTERRUPT_REG. However, if a transmit interrupt occurs whilst the INTERRUPT_REG is being read (i.e., in the same APB clock cycle), the transmit interrupt is lost.

Workarounds

When a message is awaiting completion of transmission (i.e., transmission has been requested), users should also check the STATUS_TRANSMIT_BUFFER bit each time the INTERRUPT_REG is read. A set STATUS_TRANSMIT_BUFFER bit whilst the TWAI_TRANSMIT_INT_ST is not indicates a lost transmit interrupt.

Solution

No fix scheduled.

3.32 [LEDC] When the LEDC is in decremental fade mode, a duty overflow error may occur

Affected revisions: v0.0 v1.0 v1.1 v3.0 v3.1

Description

This issue may happen when the LEDC is in decremental fade mode and LEDC_DUTY_SCALE_HSCH*n* is 1. If the duty is $2^{\text{LEDC\_HSTIMERx\_DUTY\_RES}}$, then the next one should be $2^{\text{LEDC\_HSTIMERx\_DUTY\_RES}} - 1$, however, the next duty is actually $2^{\text{LEDC\_HSTIMERx\_DUTY\_RES+1}}$, which indicates a duty overflow error. (HSCH*n* refers to high-speed channel with *n* being 0-7; HSTIMER*x* refers to high-speed timer with *x* being 0-3.)

For low-speed channels, the same issue may also happen.

Workarounds

When using LEDC, avoid the concurrence of following three cases:

1. The LEDC is in decremental fade mode;
2. The scale register is set to 1;
3. The duty is $2^{\text{LEDC\_HSTIMERx\_DUTY\_RES}}$ or $2^{\text{LEDC\_LSTIMERx\_DUTY\_RES}}$.

Solution

This issue is automatically worked around in the LEDC driver since the ESP-IDF commit ID b2e264e and is released in ESP-IDF v3.1.

4 Revision History

Table 4.1: Revision History

Date	Version	Release Notes
2024-07-29	v2.8	• Added Section *[Clock] ESP32 cannot be used as the PHY clock source if Wi-Fi and Ethernet are used at the same time*
2023-09-19	v2.7	• Added Sections *[RTC] RTC Register Read Error After Wake-up from Light-sleep Mode* and *[CPU] When a CPU is interrupted while accessing five specific FIFO registers, subsequent CPU accesses will get halted* • Updated Sections *[GPIO] Within the same group of GPIO pins, edge interrupts cannot be used together with other interrupts* and *[UART] UART fifo_cnt does not indicate the data length in FIFO correctly*
2023-02-02	v2.6	• Removed hall sensor from Section *[GPIO] When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns* according to PCN
2022-11-23	v2.5	• Added register GPIO_OUT_W1TS_REG in Section *[CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost*
2022-10-13	v2.4	• Added chip revision v3.1 and v1.1 • Added Sections *[TWAI] When the RX FIFO overruns with 64 or more messages, the RX FIFO becomes unrecoverable* and *[ULP] ULP coprocessor and touch sensors can not be used during Deep-sleep when RTC_PERIPH power domain is up* • Renamed this document as "ESP32 Series SoC Errata"
2020-09-25	v2.3	• Updated Section *[CPU] There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces* and provided more information about UART FIFO read operation
2020-06-08	v2.2	• Added Sections *[UART] UART fifo_cnt does not indicate the data length in FIFO correctly* and *[CPU] CPU has limitations when accessing peripherals in chips*
2020-05-14	v2.1	• Added a note of fix in Section *[Reset] Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deepsleep*

Date	Version	Release Notes
2020-05-08	v2.0	• Added Sections *[Watchdog] ESP32 chip may have a live lock under certain conditions that will cause interrupt watchdog issue* and *[CPU] There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces* • Added a note in Section *[CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost* • Updated the address ranges of space A and B in Section *[CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur* and fixed a typo
2020-03-16	v1.9	• Added chip revision 3 in Table Chip Revision Identification by Chip Marking • Added note of fixes in sections *[CPU] When the CPU accesses external SRAM in a certain sequence, read and write errors may occur* and *[CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur* • Added Sections *[TWAI] A negative phase error where |e| > SJW(N) will cause the remaining transmitted bits to be left shifted* and *[GPIO] Within the same group of GPIO pins, edge interrupts cannot be used together with other interrupts* • Added documentation feedback link
2018-12	v1.8	• Added Section "ESP32 TWAI Errata"
2018-05	v1.7	• Added Section *[LEDC] When the LEDC is in decremental fade mode, a duty overflow error may occur*
2018-05	v1.6	• Overall update
2018-02	v1.5	• Added Section *[GPIO] When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns*
2018-02	v1.4	• Corrected typos in the register names in Section *[CPU] When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost*

Date	Version	Release Notes
2017-06	v1.3	• Added Sections *[CPU] When the CPU accesses external SRAM in a certain sequence, read and write errors may occur* and *[CPU] When each CPU reads certain different address spaces simultaneously, a read error may occur*
2017-04	v1.2	• Changed the description of Section *[Reset] A spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep* • Added Section *[Reset] Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deepsleep*
2016-12	v1.1	• Modified the MEMW command in Section *[CPU] When the CPU accesses external SRAM through cache, under certain conditions read and write errors occur*
2016-11	v1.0	• First release

5 Related Documentation and Resources

5.1 Related Documentation

- ESP32 Datasheet –Specifications of the ESP32 hardware.
- ESP32 Technical Reference Manual –Detailed information on how to use the ESP32 memory and peripherals.
- ESP32 Hardware Design Guidelines –Guidelines on how to integrate the ESP32 into your hardware product.
- Certificates
  https://espressif.com/en/support/documents/certificates
- ESP32 Product/Process Change Notifications (PCN)
  https://espressif.com/en/support/documents/pcns?keys=ESP32
- ESP32 Advisories –Information on security, bugs, compatibility, component reliability.
  https://espressif.com/en/support/documents/advisories?keys=ESP32
- Documentation Updates and Update Notification Subscription
  https://espressif.com/en/support/download/documents

5.2 Developer Zone

- ESP-IDF Programming Guide for ESP32 –Extensive documentation for the ESP-IDF development framework.
- ESP-IDF and other development frameworks on GitHub.
  https://github.com/espressif
- ESP32 BBS Forum –Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- The ESP Journal –Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs SDKs and Demos, Apps, Tools, AT Firmware.
  https://espressif.com/en/support/download/sdks-demos

5.3 Products

- ESP32 Series SoCs –Browse through all ESP32 SoCs.
  https://espressif.com/en/products/socs?id=ESP32
- ESP32 Series Modules –Browse through all ESP32-based modules.
  https://espressif.com/en/products/modules?id=ESP32
- ESP32 Series DevKits –Browse through all ESP32-based devkits.
  https://espressif.com/en/products/devkits?id=ESP32
- ESP Product Selector –Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector

5.4 Contact Us

- See the tabs Sales Questions, Technical Enquiries, Circuit Schematic & PCB Design Review, Get Samples (Online stores), Become Our Supplier, Comments & Suggestions.
  https://espressif.com/en/contact-us/sales-questions