ESP32

ECO and Workarounds for Bugs

Version 2.2
Espressif Systems

About This Guide

This document details hardware errata and workarounds in the ESP32.

Release Notes

Date	Version	Release notes
2016-11	V1.0	Initial release.
2016-12	V1.1	Modified the MEMW command in Section 3.2.
2017-04	V1.2	Changed the description of Section 3.1; Added a bug in Section 3.8.
2017-06	V1.3	Added items 3.9 and 3.10
2018-02	V1.4	Corrected typos in the register names in Section 3.3.
2018-02	V1.5	Added Section 3.11.
2018-05	V1.6	Overall update.
2018-05	V1.7	Added Section 3.12.
2018-12	V1.8	Added Section 3.13: ESP32 CAN Errata.
2020-03-16	V1.9	• Added chip revision 3 in Table 1-1 • Added note of fixes in sections 3.9 and 3.10 • Added Section 3.13.10 • Added Section 3.14 • Added documentation feedback link
2020-05-08	V2.0	• Added Section 3.15 • Added Section 3.16 • Added a note in Section 3.3 • Updated the address ranges of space A and B in Section 3.10 and fixed a typo
2020-05-14	V2.1	Added a note of fix in Section 3.8.
2020-06-08	V2.2	Added sections 3.17 and 3.18.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe. Note that you need to update your subscription to receive notifications of new products you are not currently subscribed to.

Certification

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

Table of Contents

1. Chip Revision

The chip revision is identified by the registers and eFuse identification bits. Details can be found in the table below.

Table 1-1. Chip Revision

	Register address		
Chip revision	APB_CTRL_DATE[31]	EFUSE_BLK0_RDATA5[20]	EFUSE_BLK0_RDATA3[15]
V0, without ECO	0	0	0
ECO, V1	0	0	1
ECO, V3	1	1	1

2. Errata List

Table 2-1. Errata Summary

Section	Title	Affected revisions
Section 3.1	A spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep.	V0
Section 3.2	When the CPU accesses external SRAM through cache, under certain conditions read and write errors occur.	V0
Section 3.3	When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost.	V0
Section 3.4	The Brown-out Reset (BOR) function does not work. The system fails to boot up after BOR.	V0
Section 3.5	The CPU crashes when the clock frequency switches directly from 240 MHz to 80/160 MHz.	V0
Section 3.6	GPIO pull-up and pull-down resistors for pads with both GPIO and RTC_GPIO functionality can only be controlled via RTC_GPIO registers.	V0/V1/V3
Section 3.7	Audio PLL frequency range is limited.	V0
Section 3.8	Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep.	V0/V1
Section 3.9	When the CPU accesses external SRAM in a certain sequence, read and write errors can occur.	V1
Section 3.10	When each CPU reads certain different address spaces simultaneously, a read error can occur.	V0/V1
Section 3.11	When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns.	V0/V1/V3
Section 3.12	When the LEDC is in decremental fade mode, a duty overflow error can occur.	V0/V1/V3
Section 3.13	ESP32 CAN Errata	
Section 3.13.1	Receive Error Counter (REC) is allowed to change whilst in reset mode or bus-off recovery.	V0/V1/V3
Section 3.13.2	Error status bit is not frozen during bus-off recovery.	V0/V1/V3
Section 3.13.3	Message transmitted after bus-off recovery is erroneous.	V0/V1/V3
Section 3.13.4	Reading the interrupt register can lead to a transmit interrupt being lost.	V0/V1/V3
Section 3.13.5	Receiving an erroneous data frame can cause the data bytes of the next received data frame to be invalid.	V0/V1/V3

Section	Title	Affected revisions
Section 3.13.6	After losing arbitration, a dominant bit on the 3rd bit of intermission is not interpreted as an SOF.	V0/V1/V3
Section 3.13.7	When the 8th bit of the error delimiter is dominant, the error passive state is not entered.	V0/V1/V3
Section 3.13.8	Suspend transmission is included even after losing arbitration.	V0/V1/V3
Section 3.13.9	When a stuff error occurs during arbitration whilst being transmitter, any errors in the subsequent error/overload frame will not increase the TEC.	V0/V1/V3
Section 3.13.10	A negative phase error where |e| > SJW(N) will cause the remaining transmitted bits to be left shifted.	V0/V1/V3
Section 3.14	The ESP32 GPIO peripheral may not trigger interrupts correctly.	V0/V1/V3
Section 3.15	The ESP32 chip may have a live lock under certain conditions that will cause interrupt watchdog issue.	V3
Section 3.16	There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces.	V0/V1/V3
Section 3.17	UART fifo_cnt is inconsistent with FIFO pointer.	V0/V1/V3
Section 3.18	CPU has limitations when accessing peripherals in chips.	V0/V1/V3

3. Errata Descriptions and Workarounds

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

Workarounds:

When waking from Deep-sleep, this bug is worked around automatically in ESP-IDF V1.0 and newer.

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

Workaround Details:

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.

Fixes:

This issue is fixed in silicon revision 1.

3.2. When the CPU accesses external SRAM through cache, under certain conditions read and write errors occur.

Details:

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

Workarounds:

There is no automatic workaround available in software.

Workaround Details:

If accessing external SRAM from a revision 0 ESP32, 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.

Fixes:

This issue is fixed in silicon revision 1.

3.3. When the CPU accesses peripherals and writes a single address repeatedly, some writes may be lost.

Details:

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:

This issue is automatically worked around in the drivers of ESP-IDF V1.0 and newer.

Workaround Details:

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_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

Registers	DPORT Addresses	AHB (Safe) Addresses
GPIO_ENABLE1_W1TC_REG	0x3FF44034	0x60004034

Fixes:

This issue is fixed in silicon revision 1.

> ⚠ **Notice:**
>
> Software cannot use AHB addresses to read FIFO.

3.4. 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.

Fixes:

This issue is fixed in silicon revision 1.

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

Workarounds:

This issue is automatically worked around in ESP-IDF V2.1 and newer.

Workaround Details:

When switching frequencies, use intermediate frequencies as follows:

(1) 2 MHz <-> 40 MHz <-> 80 MHz <-> 160 MHz

(2) 2 MHz <->40 MHz <->240 MHz

Fixes:

This issue is fixed in silicon revision 1.

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

Details:

For these pads, the GPIO pull-up and pull-down configuration register fields are non-functional.

Workarounds:

This issue is automatically worked around when using GPIO drivers in ESP-IDF V2.1 or newer.

Workaround Details:

Use RTC_GPIO registers for both GPIO and RTC_GPIO functions.

3.7. Audio PLL frequency range is limited.

Details:

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 0, the Audio PLL frequency is calculated in hardware as follows:

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

For chip revision 1 onwards this bug is fixed and the Audio PLL frequency is calculated in hardware as follows:

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

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 silicon revision 0.

Fixes:

This issue is fixed in silicon revision 1.

3.8. Due to the flash start-up time, a spurious watchdog reset occurs when ESP32 is powered up or wakes up from Deep-sleep.

Details:

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 Deep-sleep.
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.

Fixes:

This issue is fixed in silicon revision 3 (ECO V3).

3.9. When the CPU accesses the external SRAM in a certain sequence, read & write errors can occur.

Details:

This error can 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.

Workaround Details:

- 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`.

Fixes:

This issue is fixed in silicon revision 3 (ECO V3).

3.10. When each CPU reads certain different address spaces simultaneously, a read error can occur.

Details:

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 can be generated on the CPU reading address space B.

Workarounds:

This issue is automatically worked around in ESP-IDF V3.0 and newer.

Workaround Details:

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).

Fixes:

This issue is fixed in silicon revision 3 (ECO V3).

3.11. When certain RTC peripherals are powered on, the inputs of GPIO36 and GPIO39 will be pulled down for approximately 80 ns.

Details:

Powering on the following RTC peripherals will trigger this issue:

- SARADC1
- SARADC2
- AMP
- HALL

Workarounds:

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

3.12. When the LEDC is in decremental fade mode, a duty overflow error can occur.

Details:

This issue may happen when the LEDC is in decremental fade mode and LEDC_DUTY_SCALE_HSCH*n* is 1. If the duty is $2^{\text{LEDC\_HSTIMER}x\text{\_DUTY\_RES}}$, then the next one

should be $2^{LEDC\_HSTIMER\textit{x}\_DUTY\_RES} - 1$, however, the next duty is actually $2^{LEDC\_HSTIMER\textit{x}\_DUTY\_RES+1}$, 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:

This issue is automatically worked around in the LEDC driver since the ESP-IDF commit ID b2e264e and will be part of the ESP-IDF V3.1 release.

Workaround Details:

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 $2^{LEDC\_HSTIMER\textit{x}\_DUTY\_RES}$ or $2^{LEDC\_LSTIMER\textit{x}\_DUTY\_RES}$.

3.13. ESP32 CAN Errata

3.13.1. Receive Error Counter (REC) is allowed to change whilst in reset mode or bus-off recovery.

Details:

When the CAN controller enters reset mode (e.g., by setting the RESET_MODE bit or due to a bus-off condition) or when the CAN 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 CAN controller should set the 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.

3.13.2. Error status bit is not frozen during bus-off recovery.

Details:

When the CAN 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.

3.13.3. Message transmitted after bus-off recovery is erroneous.

Details:

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

Workarounds:

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

3.13.4. Reading the interrupt register can lead to a transmit interrupt being lost.

Details:

The CAN 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 CAN_TRANSMIT_INT_ST is not indicates a lost transmit interrupt.

3.13.5. Receiving an erroneous data frame can cause the data bytes of the next received data frame to be invalid.

Details:

When the CAN 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 CAN 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 CAN bus.

- Hardware reset the CAN controller (will require saving and restoring the current register values).

3.13.6. After losing arbitration, a dominant bit on the 3rd bit of intermission is not interpreted as an SOF.

Details:

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 CAN controller loses arbitration and the following intermission's 3rd bit is dominant, the CAN 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.

3.13.7. When the 8th bit of the error delimiter is dominant, the error passive state is not entered.

Details:

When the CAN 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.

3.13.8. Suspend transmission is included even after losing arbitration.

Details:

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 CAN 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 CAN controller being late to start retransmission. Therefore, if another node transmits immediately after the interframe space is over, the CAN 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.

3.13.9. When a stuff error occurs during arbitration whilst being transmitter, any errors in the subsequent error/overload frame will not increase the TEC.

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 CAN controller is able to fulfill these requirements without issue.

However, errors within the subsequent error/overload frames themselves will fail to increase the CAN 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.

3.13.10. A negative phase error where |e| > SJW(N) will cause the remaining transmitted bits to be left shifted.

Details:

When the CAN 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 CAN 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.

3.14. The ESP32 GPIO peripheral may not trigger interrupts correctly.

Details:

The ESP32 GPIO peripheral may not trigger interrupts correctly if multiple GPIO pads are configured with edge-triggered interrupts.

Workaround 1:

Follow the steps below to trigger a GPIO interrupt on a rising edge:

1. Set the GPIO interrupt type to high.
2. Set the interrupt trigger type of the CPU to edge.

3. 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.

Similarly, follow the steps below to trigger a GPIO interrupt on a falling edge:

1. Set the GPIO interrupt type to low.
2. Set the interrupt trigger type of the CPU to edge.
3. 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.

Workaround 2:

Assuming GPIO0 ~ GPIO31 is Group1 and GPIO32 ~ GPIO39 is Group2.

- If an edge-triggered interrupt is configured in either group then no other GPIO interrupt of any type should be configured in the same group.
- Any number of level-triggered interrupts can be configured in a single group, if no edge-triggered interrupts are configured in that group.

3.15. The ESP32 chip may have a live lock under certain conditions that will cause interrupt watchdog issue.

Details:

On ESP32 ECO V3, 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.

3.16. There are limitations to the CPU access to 0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF address spaces.

Details:

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.

Workarounds:

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.

3.17. UART fifo_cnt is inconsistent with FIFO pointer.

Details:

When software uses DPORT to read UART fifo_cnt and FIFO pointer and such read operation is interrupted, FIFO pointer will not decrease by 1, but fifo_cnt will by mistake.

Workarounds:

When using DPORT to read FIFO, please calculate the number of bytes according to the value of FIFO pointer.

3.18. CPU has limitations when accessing peripherals in chips.

Details:

As described in Section 3.3, 3.10 and 3.16, CPU has limitations when accessing peripherals in chips of different versions using 0x3FF0_0000 ~ 0x3FF1_EFFF, 0x3FF4_0000 ~ 0x3FF7_FFFF, and 0x6000_0000 ~ 0x6003_FFFF.

Address space (Bus)	Register type	Operation	Chip Version		
			V0	V1	V3
0x3FF0_0000 ~ 0x3FF1_EFFF and 0x3FF4_0000 ~ 0x3FF7_FFFF (DPORT)	Non-FIFO	Write	✔		✔
		Read	✘ (See 3.10)		✔
	FIFO	Write	✘ (See 3.3)	✔	
		Read	✔	✔	
0x6000_0000 ~ 0x6003_FFFF (AHB)	Non-FIFO	Write	✔		
		Read	✔		
	FIFO	Write	✔		
		Read	✘ (No such feature, unpredictable results)		

Legend

✔ - operation is executed correctly

✘ - operation fails

Espressif IoT Team

www.espressif.com