STM32L4P5xx STM32L4Q5xx

Errata sheet

STM32L4P5xx/Q5xx device errata

Applicability

This document applies to the part numbers of STM32L4P5xx/Q5xx devices and the device variants as stated in this page.

It gives a summary and a description of the device errata, with respect to the device datasheet and reference manual RM0432.

Deviation of the real device behavior from the intended device behavior is considered to be a device limitation. Deviation of the description in the reference manual or the datasheet from the intended device behavior is considered to be a documentation erratum. The term *"errata"* applies both to limitations and documentation errata.

Table 1. **Device summary**

Reference	Part numbers
STM32L4P5xx	STM32L4P5ZG, STM32L4P5ZE, STM32L4P5VG, STM32L4P5VE, STM32L4P5RG, STM32L4P5RE, STM32L4P5QG, STM32L4P5QE, STM32L4P5CG, STM32L4P5CE, STM32L4P5AG, STM32L4P5AE
STM32L4Q5xx	STM32L4Q5ZG, STM32L4Q5VG, STM32L4Q5RG, STM32L4Q5QG, STM32L4Q5CG, STM32L4Q5AG

Table 2. **Device variants**

Reference	Silicon revision codes	
	Device marking(1)	REV_ID(2)
STM32L4P5xx/Q5xx	Z	0x1001

1. *Refer to the device datasheet for how to identify this code on different types of package.*
2. *REV_ID[15:0] bitfield of DBGMCU_IDCODE register.*

1 Summary of device errata

The following table gives a quick reference to the STM32L4P5xx/Q5xx device limitations and their status:

A = workaround available

N = no workaround available

P = partial workaround available

Applicability of a workaround may depend on specific conditions of target application. Adoption of a workaround may cause restrictions to target application. Workaround for a limitation is deemed partial if it only reduces the rate of occurrence and/or consequences of the limitation, or if it is fully effective for only a subset of instances on the device or in only a subset of operating modes, of the function concerned.

Table 3. Summary of device limitations

Function	Section	Limitation	Status
			Rev. Z
Core	2.1.1	Interrupted loads to SP can cause erroneous behavior	A
	2.1.2	VDIV or VSQRT instructions might not complete correctly when very short ISRs are used	A
	2.1.3	Store immediate overlapping exception return operation might vector to incorrect interrupt	A
System	2.2.1	Full JTAG configuration without NJTRST pin cannot be used	A
	2.2.2	Data cache might be corrupted during flash memory read-while-write operation	A
	2.2.3	HSE oscillator long startup at low voltage	P
	2.2.4	FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation	A
	2.2.6	SRAM write error	A
	2.2.7	Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled	A
	2.2.8	Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop	A
	2.2.9	Ports assigned to DAC cannot be used as GPIOs when the DAC feeds OPAMP	N
	2.2.10	PC13 signal transitions disturb LSE	N
DMA	2.3.1	DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear	A
DMAMUX	2.4.1	SOFx not asserted when writing into DMAMUX_CFR register	N
	2.4.2	OFx not asserted for trigger event coinciding with last DMAMUX request	N
	2.4.3	OFx not asserted when writing into DMAMUX_RGCFR register	N
	2.4.4	Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event	A
FMC	2.5.1	Dummy read cycles inserted when reading synchronous memories	N
	2.5.2	Wrong data read from a busy NAND memory	A
	2.5.3	Data corruption upon a specific FIFO write sequence to synchronous PSRAM	A
OCTOSPI	2.6.1	Spurious interrupt in AND-match polling mode with full data masking	A
	2.6.2	Odd address alignment and odd byte number not supported at specific conditions	A
	2.6.3	Data not sampled correctly on reads without DQS and with less than two cycles before the data phase	A
	2.6.4	Memory-mapped write error response when DQS output is disabled	P

Function	Section	Limitation	Status
			Rev. Z
OCTOSPI	2.6.5	Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split	A
	2.6.6	Single-, dual- and quad-SPI modes not functional with DQS input enabled	N
	2.6.7	Additional bytes read in Indirect mode with DQS input enabled when data length is too short	A
	2.6.8	Received data corrupted after arbitration ownership toggles when using clock mode 3 and no DQS for read direction	A
	2.6.9	Deadlock can occur under certain conditions	A
	2.6.10	Read data corruption after partial read when crossing CSBOUND boundary	P
	2.6.11	Deadlock or write-data corruption after spurious write to a misaligned address in OCTOSPI_ AR register	N
	2.6.12	Read data corruption after a few bytes are skipped when crossing a four-byte boundary	A
OCTOSPIM	2.7.1	Unaligned write access to OCTOSPIM configuration registers failing	A
ADC	2.8.1	New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0	A
	2.8.2	Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0	A
	2.8.3	Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode	A
	2.8.4	ADC_AWDy_OUT reset by non-guarded channels	A
	2.8.5	Wrong ADC result if conversion done late after calibration or previous conversion	A
	2.8.6	Wrong ADC differential conversion result for channel 5	A
	2.8.7	Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled	A
DAC	2.9.1	Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization	A
	2.9.2	DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge	N
COMP	2.10.1	Comparator outputs cannot be configured in open-drain	N
TSC	2.11.1	TSC signal-to-noise concern under specific conditions	A
TIM	2.13.1	One-pulse mode trigger not detected in master-slave reset + trigger configuration	P
	2.13.2	Consecutive compare event missed in specific conditions	N
	2.13.3	Output compare clear not working with external counter reset	P
	2.13.4	HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE	A
LPTIM	2.14.1	Device may remain stuck in LPTIM interrupt when entering Stop mode	A
	2.14.2	ARRM and CMPM flags are not set when APB clock is slower than kernel clock	P
	2.14.3	Device may remain stuck in LPTIM interrupt when clearing event flag	P
	2.14.4	LPTIM events and PWM output are delayed by one kernel clock cycle	P
	2.14.5	LPTIM1 outputs cannot be configured as open-drain	N
RTC and TAMP	2.15.1	Alarm flag may be repeatedly set when the core is stopped in debug	N
	2.15.2	A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF	N

Function	Section	Limitation	Status
			Rev. Z
RTC and TAMP	2.15.3	REFCKON write protection associated to INIT KEY instead of CAL KEY	A
	2.15.4	Tamper flag not set on LSE failure detection	N
	2.15.5	Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1	A
	2.15.6	RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode	P
I2C	2.16.1	Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period	P
	2.16.2	Spurious bus error detection in master mode	A
	2.16.3	OVR flag not set in underrun condition	N
	2.16.4	Transmission stalled after first byte transfer	A
USART	2.17.1	Anticipated end-of-transmission signaling in SPI slave mode	A
	2.17.2	Data corruption due to noisy receive line	N
LPUART	2.18.1	LPUART1 outputs cannot be configured as open-drain	N
SPI	2.19.1	BSY bit may stay high when SPI is disabled	A
	2.19.2	BSY bit may stay high at the end of data transfer in slave mode	A
SAI	2.20.1	Last SAI_MCLK clock pulse truncated upon disabling SAI master	A
bxCAN	2.21.1	bxCAN time-triggered communication mode not supported	N
OTG_FS	2.22.1	Host packet transmission may hang when connecting through a hub to a low-speed device	N

The following table gives a quick reference to the documentation errata.

Table 4. **Summary of device documentation errata**

Function	Section	Documentation erratum
System	2.2.5	Flash memory ECC single-error correction prevents double-error detection from triggering NMI
HASH	2.12.1	Superseded suspend sequence for data loaded by DMA
	2.12.2	Superseded suspend sequence for data loaded by the CPU

2 Description of device errata

The following sections describe the errata of the applicable devices with Arm® core and provide workarounds if available. They are grouped by device functions.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2.1 Core

Reference manual and errata notice for the Arm® Cortex®-M4 FPU core revision r0p1 is available from http://infocenter.arm.com.

2.1.1 Interrupted loads to SP can cause erroneous behavior

This limitation is registered under Arm ID number 752770 and classified into “Category B”. Its impact to the device is minor.

Description

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), erroneous behavior can occur. In all cases, returning from the interrupt will result in the load instruction being executed an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The affected instructions that can result in the load transaction being repeated are:

- LDR SP, [Rn],#imm
- LDR SP, [Rn,#imm]!
- LDR SP, [Rn,#imm]
- LDR SP, [Rn]
- LDR SP, [Rn,Rm]

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

- LDR SP,[Rn],#imm
- LDR SP,[Rn,#imm]!

As compilers do not generate these particular instructions, the limitation is only likely to occur with hand-written assembly code.

Workaround

Both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

2.1.2 VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

This limitation is registered under Arm ID number 776924 and classified into “Category B”. Its impact to the device is limited.

Description

The VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to the FPSCR.

The failure occurs when the following condition is met:

1. The floating point unit is enabled
2. Lazy context saving is not disabled
3. A VDIV or VSQRT is executed
4. The destination register for the VDIV or VSQRT is one of s0 - s15
5. An interrupt occurs and is taken
6. The interrupt service routine being executed does not contain a floating point instruction
7. Within 14 cycles after the VDIV or VSQRT is executed, an interrupt return is executed

A minimum of 12 of these 14 cycles are utilized for the context state stacking, which leaves 2 cycles for instructions inside the interrupt service routine, or 2 wait states applied to the entire stacking sequence (which means that it is not a constant wait state for every access).

In general, this means that if the memory system inserts wait states for stack transactions (that is, external memory is used for stack data), then this erratum cannot be observed.

The effect of this erratum is that the VDIV or VQSRT instruction does not complete correctly and the register bank and FPSCR are not updated, which means that these registers hold incorrect, out of date, data.

Workaround

A workaround is only required if the floating point unit is enabled. A workaround is not required if the stack is in external memory.

There are two possible workarounds:

- Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).
- Ensure that every interrupt service routine contains more than 2 instructions in addition to the exception return instruction.

2.1.3 Store immediate overlapping exception return operation might vector to incorrect interrupt

This limitation is registered under Arm ID number 838869 and classified into “Category B (rare)”. Its impact to the device is minor.

Description

The core includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, a late change in selection of the next interrupt to be taken might result in there being a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

The failure occurs when the following condition is met:

1. The handler for interrupt A is being executed.
2. Interrupt B, of the same or lower priority than interrupt A, is pending.
3. A store with immediate offset instruction is executed to a bufferable location.
   – STR/STRH/STRB <Rt>, [<Rn>,#imm]
   – STR/STRH/STRB <Rt>, [<Rn>,#imm]!
   – STR/STRH/STRB <Rt>, [<Rn>],#imm
4. Any number of additional data-processing instructions can be executed.
5. A BX instruction is executed that causes an exception return.
6. The store data has wait states applied to it such that the data is accepted at least two cycles after the BX is executed.
   – Minimally, this is two cycles if the store and the BX instruction have no additional instructions between them.
   – The number of wait states required to observe this erratum needs to be increased by the number of cycles between the store and the interrupt service routine exit instruction.
7. Before the bus accepts the buffered store data, another interrupt C is asserted which has the same or lower priority as A, but a greater priority than B.

Example:

The processor should execute interrupt handler C, and on completion of handler C should execute the handler for B. If the conditions above are met, then this erratum results in the processor erroneously clearing the pending state of interrupt C, and then executing the handler for B twice. The first time the handler for B is executed it will be at interrupt C's priority level. If interrupt C is pended by a level-based interrupt which is cleared by C's handler then interrupt C will be pended again once the handler for B has completed and the handler for C will be executed.

As the STM32 interrupt C is level based, it eventually becomes pending again and is subsequently handled.

Workaround

For software not using the memory protection unit, this erratum can be worked around by setting DISDEFWBUF in the Auxiliary Control Register.

In all other cases, the erratum can be avoided by ensuring a DSB occurs between the store and the BX instruction. For exception handlers written in C, this can be achieved by inserting the appropriate set of intrinsics or inline assembly just before the end of the interrupt function, for example:

ARMCC:

```
...
__schedule_barrier();
__asm{DSB};
__schedule_barrier();
}
```

GCC:

```
...
__asm volatile ("dsb 0xf":::"memory");
}
```

2.2 System

2.2.1 Full JTAG configuration without NJTRST pin cannot be used

Description

When using the JTAG debug port in Debug mode, the connection with the debugger is lost if the NJTRST pin (PB4) is used as a GPIO or for an alternate function other than NJTRST. Only the 4-wire JTAG port configuration is impacted.

Workaround

Use the SWD debug port instead of the full 4-wire JTAG port.

2.2.2 Data cache might be corrupted during flash memory read-while-write operation

Description

When a write to the internal flash memory is done, the data cache is normally updated to reflect the data value update. During this data cache update, a read to the other flash memory bank may occur; this read can corrupt the data cache content and subsequent read operations at the same address (cache hits) will be corrupted.

This limitation only occurs in dual bank mode, when reading (data access or code execution) from one bank while writing to the other bank with data cache enabled.

Workaround

When the application is performing data accesses in both flash memory banks, the data cache must be disabled by resetting the DCEN bit before any write to the flash memory. Before enabling the data cache again, it must be reset by setting and then resetting the DCRST bit.

Code example:

```
/* Disable data cache  */
__HAL_FLASH_DATA_CACHE_DISABLE();

/* Set PG bit */
SET_BIT(FLASH->CR, FLASH_CR_PG);

/* Program the Flash word */
WriteFlash(Address, Data);

/* Reset data cache */
__HAL_FLASH_DATA_CACHE_RESET();

/* Enable data cache */
__HAL_FLASH_DATA_CACHE_ENABLE();
```

2.2.3 HSE oscillator long startup at low voltage

Description

When V_{DD} is below 2.7 V, the HSE oscillator may take longer than specified to start up. Several hundred milliseconds might elapse before the HSERDY flag in the RCC_CR register is set.

Workaround

The following sequence is recommended:

1. Configure PH0 and PH1 as standard GPIOs in output mode and low-level state.
2. Enable the HSE oscillator.

2.2.4 FLASH_ECCR corrupted upon reset or power-down occurring during flash memory program or erase operation

Description

Reset or power-down occurring during a flash memory location program or erase operation, followed by a read of the same memory location, may lead to a corruption of the FLASH_ECCR register content.

Workaround

Under such condition, erase the page(s) corresponding to the flash memory location.

2.2.5 Flash memory ECC single-error correction prevents double-error detection from triggering NMI

Description

Some reference manual revisions may not state that:

- A double-error detection event occurring while the ECCC (or ECCC2) flag is set (after a single-error correction event) does not generate NMI.
- It is recommended to clear the ECCC (or ECCC2) flag as soon as possible, to preserve the ECC error detection capability.

This is a documentation issue rather than a device limitation.

Workaround

No application workaround is applicable provided that the recommendation is respected.

2.2.6 SRAM write error

Description

In rare cases, system reset occurring in a critical instant may upset the SRAM state machine. The first SRAM read or write access after the system reset then restores the normal operation of the SRAM for any subsequent accesses. However, if it is a write access, it fails to write data.

Workaround

Upon system reset, make a dummy read access to each 32-Kbyte SRAM instance used by the application, through one of the following methods:

1. Place a dummy variable initialization, which allows the compiler to create the assembly code.
2. Use this initialization assembly code:

```
MOV32 R0, #0x20000000 //SRAM address
LDR R1, [R0, #+0] //read access
MOV32 R0, #0x20008000 //next SRAM instance
LDR R1, [R0, #+0] //dummy read, no consequence on R1 value
...
MOV32 R0, #0x20020000 //first SRAM2 cut
LDR R1, [R0, #+0]
...
```

Follow the first method for SRAM instances with the parity check enabled. Follow the first or the second method for SRAM instances with the parity check disabled.

2.2.7 Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled

Description

When debug in low-power mode is enabled, wrong instructions can be fetched from flash memory and executed after waking up from Sleep or Stop mode, causing an unpredictable behavior of the device or a CPU exception.

This issue occurs when the device exits Sleep or Stop mode in the following conditions:

- At least one of DBG_SLEEP or DBG_STOP bit of DBGMCU_CR register is set.
- Interrupts with wakeup capability are disabled at Sleep or Stop entry.
- The flash memory interface gating is enabled by clearing FLASHSMEN bit of RCC_AHB1SMENR (only for Sleep mode).

Note: The issue affects only debug mode and has no effect on real applications.

Workaround

Apply one of the following measures:

- Add an ISB just after WFI (or WFE) instruction.
- Disable the flash memory interface gating, by setting FLASHSMEN bit (valid only for Sleep mode).

2.2.8 Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop

Description

The backup domain reset may be missed upon a power-on following a power-off, if its supply voltage drops during the power-off phase hitting a window, which is few mV wide before it starts to rise again. In this critical window, the flip-flops are no longer able to safely retain the information and the backup domain reset has not yet been triggered. This window is located in the range between 100 mV and 700 mV, with the exact position depending mainly on the device and on the temperature.

This missed reset results in unpredictable values of the backup domain registers. This may cause a spurious behavior (such as driving the LSCO output pin on PA2 or influencing backup functions).

Workaround

Apply one of the following measures:

- In the application, let the V_{DD} and V_{BAT} supply voltages fall to a level below 100 mV for more than 200 ms before a new power-on.
- If the above workaround cannot be applied, and the boot follows a power-on reset, erase the backup domain by software.
  When the application is using shutdown mode, user needs to discriminate between the power-on reset or an exit from a shutdown mode.
  For this purpose, at least one backup register must have been previously programmed with a BKP_REG_VAL value with 16 bits set and 16 bits cleared.
  Robustness of this workaround can be significantly improved by using a CRC rather than registers. The registers are subject to backup domain reset.
  The workaround consists of calculating the CRC of the backup registers: RCC_BDCR and RTC registers, excluding bits modified by HW.
  The CRC result can be stored in the backup register instead of a fixed value. This value needs to be updated for each modification of values covered by CRC, such as by using CRC peripheral.
  At the very beginning of the boot code, insert the following software sequence:
  1. Check the BORRSTF flag of the RCC_CSR register. If set, the reset is caused by a power on, or is exiting from shutdown mode.
  2. If BORRSTF flag is true, and the shutdown mode is used in the application, check that the backup register value is different from BKP_REG_VAL. When tamper detection is enabled,check that no tamper flag is set. If both conditions are met then the reset is caused by a power-on.
  3. If the reset is caused by a power-on, apply the following sequence:
     a. Enable the PWR clock in the RCC, by setting the PWREN bit.
     b. Enable the backup domain access in the PWR, by setting the DBP bit.
     c. Reset the backup domain, by:
        i. Writing 0x0001 0000 in the RCC_BDCR register, which sets the BDRST bit and clears other register bits that might not be reset.
        ii. reading the RCC_BDCR register, to make the reset time long enough
        iii. writing 0x0000 0000 in the RCC_BDCR register, to clear the BDRST bit
     d. Clear the BORRSTF flag by setting the RMVF bit of the RCC_CSR register.

2.2.9 Ports assigned to DAC cannot be used as GPIOs when the DAC feeds OPAMP

Description

When a DAC output is exclusively connected to an on-chip peripheral, ports assigned to the DAC output are expected to be usable as GPIOs.

However, when the dac1_out1 and dac1_out2 DAC output signals are connected to opamp1_vinp and opamp2_vinp OPAMP inputs, respectively, the PA4 and PA5 ports assigned to the DAC outputs 1 and 2 cannot be configured as GPIOs. Instead, they must be set in analog mode.

Workaround

None.

2.2.10 PC13 signal transitions disturb LSE

Description

The PC13 port toggling disturbs the LSE clock.

Workaround

None.

2.3 DMA

2.3.1 DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear

Description

Upon a data transfer error in a DMA channel x, both the specific TEIFx and the global GIFx flags are raised and the channel x is normally automatically disabled. However, if in the same clock cycle the software clears the GIFx flag (by setting the CGIFx bit of the DMA_IFCR register), the automatic channel disable fails and the TEIFx flag is not raised.

This issue does not occur with ST's HAL software that does not use and clear the GIFx flag when the channel is active.

Workaround

Do not clear GIFx flags when the channel is active. Instead, use HTIFx, TCIFx, and TEIFx specific event flags and their corresponding clear bits.

2.4 DMAMUX

2.4.1 SOFx not asserted when writing into DMAMUX_CFR register

Description

The SOFx flag of the DMAMUX_CSR status register is not asserted if overrun from another DMAMUX channel occurs when the software writes into the DMAMUX_CFR register.

This can happen when multiple DMA channels operate in synchronization mode, and when overrun can occur from more than one channel. As the SOFx flag clear requires a write into the DMAMUX_CFR register (to set the corresponding CSOFx bit), overrun occurring from another DMAMUX channel operating during that write operation fails to raise its corresponding SOFx flag.

Workaround

None. Avoid the use of synchronization mode for concurrent DMAMUX channels, if at least two of them potentially generate synchronization overrun.

2.4.2 OFx not asserted for trigger event coinciding with last DMAMUX request

Description

In the DMAMUX request generator, a trigger event detected in a critical instant of the last-generated DMAMUX request being served by the DMA controller does not assert the corresponding trigger overrun flag OFx. The critical instant is the clock cycle at the very end of the trigger overrun condition.

Additionally, upon the following trigger event, one single DMA request is issued by the DMAMUX request generator, regardless of the programmed number of DMA requests to generate.

The failure only occurs if the number of requests to generate is set to more than two (GNBREQ[4:0] > 00001).

Workaround

Make the trigger period longer than the duration required for serving the programmed number of DMA requests, so as to avoid the trigger overrun condition from occurring on the very last DMA data transfer.

2.4.3 OFx not asserted when writing into DMAMUX_RGCFR register

Description

The OFx flag of the DMAMUX_RGSR status register is not asserted if an overrun from another DMAMUX request generator channel occurs when the software writes into the DMAMUX_RGCFR register. This can happen when multiple DMA channels operate with the DMAMUX request generator, and when an overrun can occur from more than one request generator channel. As the OFx flag clear requires a write into the DMAMUX_RGCFR register (to set the corresponding COFx bit), an overrun occurring in another DMAMUX channel operating with another request generator channel during that write operation fails to raise the corresponding OFx flag.

Workaround

None. Avoid the use of request generator mode for concurrent DMAMUX channels, if at least two channels are potentially generating a request generator overrun.

2.4.4 Wrong input DMA request routed upon specific DMAMUX_CxCR register write coinciding with synchronization event

Description

If a write access into the DMAMUX_CxCR register having the SE bit at zero and SPOL[1:0] bitfield at a value other than 00:

- sets the SE bit (enables synchronization),
- modifies the values of the DMAREQ_ID[5:0] and SYNC_ID[4:0] bitfields, and
- does not modify the SPOL[1:0] bitfield,

and if a synchronization event occurs on the previously selected synchronization input exactly two AHB clock cycles before this DMAMUX_CxCR write, then the input DMA request selected by the DMAREQ_ID[5:0] value before that write is routed.

Workaround

Ensure that the SPOL[1:0] bitfield is at 00 whenever the SE bit is 0. When enabling synchronization by setting the SE bit, always set the SPOL[1:0] bitfield to a value other than 00 with the same write operation into the DMAMUX_CxCR register.

2.5 FMC

2.5.1 Dummy read cycles inserted when reading synchronous memories

Description

When performing a burst read access from a synchronous memory, two dummy read accesses are performed at the end of the burst cycle whatever the type of burst access.
The extra data values read are not used by the FMC and there is no functional failure.

Workaround

None.

2.5.2 Wrong data read from a busy NAND memory

Description

When a read command is issued to the NAND memory, the R/B signal gets activated upon the de-assertion of the chip select. If a read transaction is pending, the NAND controller might not detect the R/B signal (connected to NWAIT) previously asserted and sample a wrong data. This problem occurs only when the MEMSET timing is configured to 0x00 or when ATTHOLD timing is configured to 0x00 or 0x01.

Workaround

Either configure MEMSET timing to a value greater than 0x00 or ATTHOLD timing to a value greater than 0x01.

2.5.3 Data corruption upon a specific FIFO write sequence to synchronous PSRAM

Description

The following specific succession of events may cause the FMC, with the write FIFO buffer enabled, to send corrupted data to a synchronous PSRAM:

1. The application software sends a data write burst to the FMC that places the data onto consecutive write FIFO buffer locations.
2. A write FIFO buffer address roll-over occurs (upon the write burst in point 1 or upon some subsequent writes to the FMC).
3. The application software writes a data byte to the FMC. The data byte reaches the same write FIFO buffer location as the first byte of the data write burst under point 1.
4. The application software writes data of any size to the FMC while the FMC is sending the data byte from point 3 to a synchronous PSRAM.

Under these circumstances, the data byte from point 3 (being sent out to PSRAM in point 4) is corrupted, or, alternatively, the data byte immediately following that data byte is corrupted.

Workaround

Use the FMC peripheral in Asynchronous write mode or disable the write FIFO buffer.

2.6 OCTOSPI

2.6.1 Spurious interrupt in AND-match polling mode with full data masking

Description

In AND-match polling mode with the MASK[31:0] bitfield set to 0x0000 0000 (all bits masked), a spurious interrupt may occur.

Workaround

Avoid setting the MASK[31:0] bitfield to 0x0000 0000.

2.6.2 Odd address alignment and odd byte number not supported at specific conditions

Description

Odd address alignment and odd transaction byte number are not supported for some combinations of memory access mode, access type, and other settings. The following table summarizes the supported combinations, and provides information on consequences of accessing an illegal address and/or of setting an illegal number of bytes in a transaction.

Table 5. **Summary of supported combinations**

Memory access mode / other settings(1)	Access type(2)	Address allowed	Consequence of illegal address access(3)	Byte number allowed	Consequence of illegal byte number(3)
Single-SPI, dual-SPI, quad-SPI, RAM / DQM = 0 or octal-SPI/SDR mode	ind read	any	N/A	any	N/A
	mm read	any	N/A	any	N/A
	ind write	any	N/A	any	N/A
	mm write	any	N/A	any	N/A
Single-SPI, dual-SPI, quad-SPI, RAM / DQM = 1 or octal-SPI, RAM / DTR mode, no RDS, no WDM	ind read	even	ADDR[0] cleared	even	DLR[0] cleared
	mm read	any	N/A	any	N/A
	ind write	even	ADDR[0] cleared	even	DLR[0] cleared
	mm write	even	slave error	even	last byte lost
Octal-SPI, RAM / DTR mode, with RDS or WDM or HyperBus™	ind read	even	ADDR[0] cleared	even	DLR[0] cleared
	mm read	any	N/A	any	N/A
	ind write	any	N/A	any	N/A
	mm write	any	N/A	any	N/A

1. *"RDS" = read data strobe, "WDM" = write data mask*
2. *"ind read" = indirect read, "mm read" = memory-mapped read, "ind write" = indirect write, "mm write" = memory-mapped write*
3. *"N/A" = not applicable*

Workaround

Avoid illegal address accesses and illegal byte numbers in transactions.

2.6.3 Data not sampled correctly on reads without DQS and with less than two cycles before the data phase

Description

A command is composed of five phases:

- Command
- Address
- Alternate byte
- Dummy (latency) cycles
- Data

Data are not sampled correctly if all the following conditions are met:

- Fewer than two cycles are required by the first four phases (command, address, alternate or dummy).
- DQS is disabled (DQSE = 0).
- Data phase is enabled.
- Data are read in Indirect or Memory-mapped mode.

Workaround

Ensure that there are at least two cycles before the data phase using one of the following methods :

- Send one byte of address in SDR quad-SPI mode (ADMODE = 011, ADSIZE = 00, ADDDTR = 0)
- Send two bytes of address in SDR octal-SPI mode (ADMODE = 100, ADSIZE = 01, ADDDTR = 0)
- Send four bytes of address in DTR octal-SPI mode (ADMODE = 100, ADSIZE = 11, ADDDTR = 1)
- Send two bytes of instruction in DTR quad-SPI mode (IMODE = 011, ISIZE = 01, IDDTR = 1)
- Send one instruction byte in octal followed by one dummy cycle.
- Send one instruction byte in octal followed by one alternate byte in octal.

2.6.4 Memory-mapped write error response when DQS output is disabled

Description

If the DQSE control bit of the OCTOSPI_WCCR register is cleared for memories without DQS pin, it results in an error response for every memory-mapped write request.

Workaround

When doing memory-mapped writes, set the DQSE bit of the OCTOSPI_WCCR register, even for memories that have no DQS pin.

2.6.5 Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split

Description

When reading a continuous stream of data from sequential addresses in a serial memory, the OCTOSPI can interrupt the transfer and automatically restart it at the next address when features generating transfer splits (CSBOUND, REFRESH, TIMEOUT or MAXTRAN) are active. Thus, a single continuous transfer can effectively be split into multiple smaller transfers.

When the OCTOSPI is configured to use clock mode 3 (CKMODE bit of the OCTOSPI_DCR1 register set) and a continuous stream of data is read in SDR mode (DDTR bit of the OCTOSPI_CCR register cleared), the last byte sent by the memory before an automatic split gets dropped, thus causing all the subsequent bytes to be seen one address earlier.

Workaround

Use clock mode 0 (CKMODE bit of the OCTOSPI_DCR1 register cleared) when in SDR mode.

2.6.6 Single-, dual- and quad-SPI modes not functional with DQS input enabled

Description

Data read from memory in single-, dual-, or quad-SPI mode with the DQS input enabled (DQSE control bit of the OCTOSPI_CCR register set) can be corrupted. Only the octal-SPI mode (DMODE bit of the OCTOSPI_CCR register set to 100) is functional with the DQS input enabled.

Workaround

None.

2.6.7 Additional bytes read in Indirect mode with DQS input enabled when data length is too short

Description

Extra byte reception may appear when the two conditions below are met at the same time:

- Data read in Indirect-read mode with DQS enabled (DQSE bit of the OCTOSPI_CCR register set)
- The number of cycles for data read phase is less than the sum of the number of cycles required for (command + address + alternate-byte + dummy) phases.

Workaround

- Avoid programming transfers with data phase shorter than (command + address + alternate-byte + dummy) phases.
- Perform an abort just after reading all the data required bytes from the OCTOSPI_DR register.

2.6.8 Received data corrupted after arbitration ownership toggles when using clock mode 3 and no DQS for read direction

Description

When two OCTOSPI peripherals compete the ownership for the same external bus through the I/O manager and the following conditions are met:

- at least one of the OCTOSPIs operating in read mode
- at least one of the OCTOSPIs set with clock mode 3

the received data is corrupted due to a spurious sampling event when the ownership of the external bus toggles.

Workaround

Use clock mode 0, by setting the bit CKMODE of the OCTOSPI_DCR1 register (all memories known to date support clock mode 0).

2.6.9 Deadlock can occur under certain conditions

Description

A deadlock can occur when all the following conditions are met:

- The product communicates through an I/O manager in multiplexed mode with an single external memory or an external combo featuring two memories, directly or through a high-speed interface.
- The external memory(ies) is(are) accessed in indirect mode or memory-mapped mode.

The deadlock can happen when the two following conditions occur at the same time:

- The Octo-SPI interface that currently owns the external bus (for example OCTOSPI1) waits for a transfer to occur with the external memory, to complete its transfer on the internal interconnect matrix bus.
- A data transfer request on the internal interconnect matrix bus arrives to the other Octo-SPI interface (for example OCTOSPI2).

This leads to an ownership conflict where:

- OCTOSPI2 cannot get ownership of the external bus which is currently in use by OCTOSPI1.
- OCTOSPI1 cannot get ownership of the internal interconnect matrix bus which is currently in use by OCTOSPI2.

Workaround

Apply one of the following measures:

- If any of the features generating automatic transfer split (MAXTRAN, REFRESH, CSBOUND, TIMEOUT) is set, OCTOSPI1 splits its transfer at some point in time, releasing the bus. OCTOSPI2 can then process its data, and when OCTOSPI1 gets ownership back again, it resumes its transfer thanks to its embedded capability to restart at the address following the last address accessed. In this case, the deadlock is resolved.
  Limitation of the workaround: The automatic resume of the transfer does not work with certain flash memories in write direction only. These memories require an extra "write enable" command before resuming a write transfer. This "write enable" command is not generated by the OCTOSPI.
- The application must ensure that it has sufficient room left in the OCTOSPI internal FIFO for each and every transfer before launching it. The internal interconnect matrix bus activity no longer depends on what happens on external bus side, and the deadlock condition is avoided.

2.6.10 Read data corruption after partial read when crossing CSBOUND boundary

Description

If a first read access of 8 or 16 bits is performed within the last 4-byte word located just before page boundary (defined in CSBOUND[4:0] of OCTOSPI_CR register), and another read operation addresses the first word of the next page while the first read operation is still ongoing on memory side, then the second read returns invalid data.

Workaround

To avoid data corruption, perform only 32-bit read accesses to the OCTOSPI memory.

For system DMA, use only 32-bit data size for read accesses.

2.6.11 Deadlock or write-data corruption after spurious write to a misaligned address in OCTOSPI_ AR register

Description

Upon writing a misaligned address to OCTOSPI_AR just before switching to Memory-mapped mode (without first triggering the indirect write operation), with the OCTOSPI configured as follows:

- FMODE = 00 in OCTOSPI_CR (Indirect write mode)
- DQSE = 1 in OCTOSPI_CCR (DQS active)

then, the OCTOSPI may be deadlocked on the first memory-mapped request or the first memory-mapped write to memory (and any sequential writes after it) may be corrupted.

An address is misaligned if:

- the address is odd and the OCTOSPI is configured to send two bytes of data to the memory every cycle (octal-DTR mode or dual-quad-DTR mode), or
- the address is not a multiple of four when the OCTOSPI is configured to send four bytes of data to the memory (16-bit DTR mode or dual-octal DTR mode).

If the OCTOSPI_AR register is reprogrammed with an aligned address (without triggering the indirect write between the two writes to OCTOSPI register), the data sent to the memory during the indirect write operation are also corrupted.

Workaround

None.

2.6.12 Read data corruption after a few bytes are skipped when crossing a four-byte boundary

Description

A memory-mapped read is corrupted when the following sequence occurs:

1. An 8- or 16-bit read is performed in a four-byte window, and the last byte of this window is not read.
2. Any read in the next four-byte window is done before the completion of the previous read memory prefetch.

OCTOSPI immediately responds to this read request, even before the data are read from memory, thus resulting in incorrect data read.

If the next read operations are issued to consecutive addresses, then the read data are also corrupted.

Workaround

Apply one of the following measures:

- Perform only 32-bit memory-mapped read accesses.
  For system DMA, use only 32-bit data size for read accesses.
- If this is not possible and an 8-bit or 16-bit read is done at the beginning of a four-byte window, ensure that the next access is not a read from the next four-byte window or that the second access occurs after the data at the skipped addresses are prefetched from memory.

2.7 OCTOSPIM

2.7.1 Unaligned write access to OCTOSPIM configuration registers failing

Description

Unaligned write access to OCTOSPIM configuration registers is discarded, with hard fault. The de-assertion of *hready* AHB signal then takes three cycles, which is not compliant with the AHB standard that defines two cycles.

Workaround

Avoid unaligned write accesses to OCTOSPIM configuration registers.

2.8 ADC

2.8.1 New context conversion initiated without waiting for trigger when writing new context in ADC_JSQR with JQDIS = 0 and JQM = 0

Description

Once an injected conversion sequence is complete, the queue is consumed and the context changes according to the new ADC_JSQR parameters stored in the queue. This new context is applied for the next injected sequence of conversions.

However, the programming of the new context in ADC_JSQR (change of injected trigger selection and/or trigger polarity) may launch the execution of this context without waiting for the trigger if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the injected conversion sequence is complete and no conversion from previous context is ongoing

Workaround

Apply one of the following measures:

- Ignore the first conversion.
- Use a queue of context with JQM = 1.
- Use a queue of context with JQM = 0, only change the conversion sequence but never the trigger selection and the polarity.

2.8.2 Two consecutive context conversions fail when writing new context in ADC_JSQR just after previous context completion with JQDIS = 0 and JQM = 0

Description

When an injected conversion sequence is complete and the queue is consumed, writing a new context in ADC_JSQR just after the completion of the previous context and with a length longer that the previous context, may cause both contexts to fail. The two contexts are considered as one single context. As an example, if the first context contains element 1 and the second context elements 2 and 3, the first context is consumed followed by elements 2 and 3 and element 1 is not executed.

This issue may happen if:

- the queue of context is enabled (JQDIS cleared to 0 in ADC_CFGR), and
- the queue is never empty (JQM cleared to 0 in ADC_CFGR), and
- the length of the new context is longer than the previous one

Workaround

If possible, synchronize the writing of the new context with the reception of the new trigger.

2.8.3 Unexpected regular conversion when two consecutive injected conversions are performed in Dual interleaved mode

Description

In Dual ADC mode, an unexpected regular conversion may start at the end of the second injected conversion without a regular trigger being received, if the second injected conversion starts exactly at the same time than the end of the first injected conversion. This issue may happen in the following conditions:

- two consecutive injected conversions performed in Interleaved simultaneous mode (DUAL[4:0] of ADC_CCR = 0b00011), or
- two consecutive injected conversions from master or slave ADC performed in Interleaved mode (DUAL[4:0]of ADC_CCR = 0b00111)

Workaround

- In Interleaved simultaneous injected mode: make sure the time between two injected conversion triggers is longer than the injected conversion time.
- In Interleaved only mode: perform injected conversions from one single ADC (master or slave), making sure the time between two injected triggers is longer than the injected conversion time.

2.8.4 ADC_AWDy_OUT reset by non-guarded channels

Description

ADC_AWDy_OUT is set when a guarded conversion of a regular or injected channel is outside the programmed thresholds. It is reset after the end of the next guarded conversion that is inside the programmed thresholds.

However, the ADC_AWDy_OUT signal is also reset at the end of conversion of non-guarded channels, both regular and injected.

Workaround

When ADC_AWDy_OUT is enabled, it is recommended to use only the ADC channels that are guarded by a watchdog.

If ADC_AWDy_OUT is used with ADC channels that are not guarded by a watchdog, take only ADC_AWDy_OUT rising edge into account.

2.8.5 Wrong ADC result if conversion done late after calibration or previous conversion

Description

The result of an ADC conversion done more than 1 ms later than the previous ADC conversion or ADC calibration might be incorrect.

Workaround

Perform two consecutive ADC conversions in single, scan or continuous mode. Reject the result of the first conversion and only keep the result of the second.

2.8.6 Wrong ADC differential conversion result for channel 5

Description

The ADC (ADC1 or ADC2) configured in differential mode with the channel 5 selected provides wrong conversion result.

Workaround

Set an unused SQxx[4:0] bitfield of the corresponding ADC_SQRx register, or an unused JSQxx[4:0] bitfield of the ADC_JSQR register, to channel 6.
For example, write the SQ16[4:0] bitfield of the ADC_SQR4 register with 00110 when the L[3:0] bitfield of the ADC_SQR1 register is set to 0001.

2.8.7 Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled

Description

When all analog peripherals are disabled, the GPIO(s) selected as ADC input(s) are unduly clamped (through a parasitic diode) to V_{DD} instead to V_{DDA}. As a consequence, the input voltage is limited to V_{DD} + 0.3 V even if V_{DDA} is higher than V_{DD} + 0.3 V.

Note: *The selection of GPIOs as ADC inputs is done with the SQy and JSQy bitfields of the ADC_SQRx and ADC_JSQR registers, respectively.*

Workaround

Apply one of the following measures:
- Use V_{DDA} lower than V_{DD} + 0.3 V.
- Keep at least one analog peripheral enabled if GPIOs are selected as ADC inputs.
- Deselect GPIOs as ADC inputs (by clearing ADC_SQRx or/and ADC_JSQR registers) when no analog peripheral is enabled.

2.9 DAC

2.9.1 Invalid DAC channel analog output if the DAC channel MODE bitfield is programmed before DAC initialization

Description

When the DAC operates in Normal mode and the DAC enable bit is cleared, writing a value different from 000 to the DAC channel MODE bitfield of the DAC_MCR register before performing data initialization causes the corresponding DAC channel analog output to be invalid.

Workaround

Apply the following sequence:
1. Perform one write access to any data register.
2. Program the MODE bitfield of the DAC_MCR register.

2.9.2 DMA underrun flag not set when an internal trigger is detected on the clock cycle of the DMA request acknowledge

Description

When the DAC channel operates in DMA mode (DMAEN of DAC_CR register set), the DMA channel underrun flag (DMAUDR of DAC_SR register) fails to rise upon an internal trigger detection if that detection occurs during the same clock cycle as a DMA request acknowledge. As a result, the user application is not informed that an underrun error occurred.

This issue occurs when software and hardware triggers are used concurrently to trigger DMA transfers.

Workaround

None.

2.10 COMP

2.10.1 Comparator outputs cannot be configured in open-drain

Description

Comparator outputs are always forced in push-pull mode whatever the GPIO output type configuration bit value.

Workaround

None.

2.11 TSC

2.11.1 TSC signal-to-noise concern under specific conditions

Description

V_{DD} equal to or greater than V_{DDA} may lead (depending on part) to some degradation of the signal-to-noise ratio on the TSC analog I/O group 2.

The lower are the sampling capacitor (C_S) and the sensing electrode (C_X) capacitances, the worse is the signal-to-noise ratio degradation.

Workaround

Apply one of the following measures:

- Maximize C_S capacitance.
- Use the analog I/O group 2 as active shield.

2.12 HASH

2.12.1 Superseded suspend sequence for data loaded by DMA

Description

The section *HASH / Context swapping / Data loaded by DMA / Current context saving* of some reference manual revisions may suggest the following suspend sequence for using HASH with DMA:

1. Clear the DMAE bit to disable the DMA interface.
2. Wait until the current DMA transfer is complete (wait for DMAS = 0 in the HASH_SR register).

This recommendation is obsolete and superseded with the following sequence that suspends then resumes the secure digest computing in order to swap the context:

Suspend:

1. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 2.
2. In Polling mode, wait for BUSY = 1.
3. Disable the DMA channel. Then clear the DMAE bit of the HASH_CR register.
4. In Polling mode, wait for BUSY = 0. If the DCIS bit of the HASH_SR register is set, the hash result is available and the context swapping is useless. Otherwise, go to step 5.
5. Save the HASH_IMR, HASH_STR, HASH_CR, and HASH_CSR0 to HASH_CSR37 registers. The HASH_CSR38 to HASH_CSR53 registers must also be saved if an HMAC operation is ongoing.

Resume:

1. Reconfigure the DMA controller so that it proceeds with the transfer of the message up to the end if it is not interrupted again. Do not forget to take into account the words already pushed into the FIFO if NBW[3:0] is higher than 0x0.
2. Program the values saved in memory to the HASH_IMR, HASH_STR, and HASH_CR registers.
3. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
4. Program the values saved in memory to the HASH_CSRx registers.
5. Restart the processing from the point of interruption, by setting the DMAE bit.

Note: To optimize the resume process when NBW[3:0] = 0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.12.2 Superseded suspend sequence for data loaded by the CPU

Description

The section *HASH / Context swapping / Data loaded by software* of some reference manual revisions may instruct that *"the user application must wait until DINIS ≠ 1 (last block processed and input FIFO empty) or NBW 0 (FIFO not full and no processing ongoing)"*.

This instruction is obsolete and superseded with the following:

When the DMA is not used to load the message into the hash processor, the context can be saved only when no block processing is ongoing.

To suspend the processing of a message, proceed as follows after writing 16 words 32-bit (plus one if it is the first block):

1. In Polling mode, wait for BUSY = 0, then poll if the DINIS status bit is set to 1. In Interrupt mode, implement the next step in DINIS interrupt handler (recommended).
2. Store the contents of the following registers into memory:
   – HASH_IMR
   – HASH_STR
   – HASH_CR
   – HASH_CSR0 to HASH_CSR37 and, if an HMAC operation is ongoing, also HASH_CSR38 to HASH_CSR53

To resume the processing of a message, proceed as follows:

1. Write the HASH_IMR, HASH_STR, and HASH_CR registers with the values saved in memory.
2. Initialize the hash processor by setting the INIT bit of the HASH_CR register.
3. Write the HASH_CSRx registers with the values saved in memory.
4. Restart the processing from the point of interruption.

Note: To optimize the resume process when NBW[3:0]=0x0, HASH_CSR22 to HASH_CSR37 registers do not need to be saved then restored as the FIFO is empty.

This is a documentation issue rather than a product limitation.

Workaround

No application workaround is required as long as the new sequence is applied.

2.13 TIM

2.13.1 One-pulse mode trigger not detected in master-slave reset + trigger configuration

Description

The failure occurs when several timers configured in one-pulse mode are cascaded, and the master timer is configured in combined reset + trigger mode with the MSM bit set:

OPM = 1 in TIMx_CR1, SMS[3:0] = 1000 and MSM = 1 in TIMx_SMCR.

The MSM delays the reaction of the master timer to the trigger event, so as to have the slave timers cycle-accurately synchronized.

If the trigger arrives when the counter value is equal to the period value set in the TIMx_ARR register, the one-pulse mode of the master timer does not work and no pulse is generated on the output.

Workaround

None. However, unless a cycle-level synchronization is mandatory, it is advised to keep the MSM bit reset, in which case the problem is not present. The MSM = 0 configuration also allows decreasing the timer latency to external trigger events.

2.13.2 Consecutive compare event missed in specific conditions

Description

Every match of the counter (CNT) value with the compare register (CCR) value is expected to trigger a compare event. However, if such matches occur in two consecutive counter clock cycles (as consequence of the CCR value change between the two cycles), the second compare event is missed for the following CCR value changes:

- in edge-aligned mode, from ARR to 0:
  - first compare event: CNT = CCR = ARR
  - second (missed) compare event: CNT = CCR = 0
- in center-aligned mode while up-counting, from ARR-1 to ARR (possibly a new ARR value if the period is also changed) at the crest (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = (ARR-1)
  - second (missed) compare event: CNT = CCR = ARR
- in center-aligned mode while down-counting, from 1 to 0 at the valley (that is, when TIMx_RCR = 0):
  - first compare event: CNT = CCR = 1
  - second (missed) compare event: CNT = CCR = 0

This typically corresponds to an abrupt change of compare value aiming at creating a timer clock single-cycle-wide pulse in toggle mode.

As a consequence:

- In toggle mode, the output only toggles once per counter period (squared waveform), whereas it is expected to toggle twice within two consecutive counter cycles (and so exhibit a short pulse per counter period).
- In center mode, the compare interrupt flag does note rise and the interrupt is not generated.

Note: The timer output operates as expected in modes other than the toggle mode.

Workaround

None.

2.13.3 Output compare clear not working with external counter reset

Description

The output compare clear event (ocref_clr) is not correctly generated when the timer is configured in the following slave modes: Reset mode, Combined reset + trigger mode, and Combined gated + reset mode.

The PWM output remains inactive during one extra PWM cycle if the following sequence occurs:

1. The output is cleared by the ocref_clr event.
2. The timer reset occurs before the programmed compare event.

Workaround

Apply one of the following measures:

- Use BKIN (or BKIN2 if available) input for clearing the output, selecting the Automatic output enable mode (AOE = 1).
- Mask the timer reset during the PWM ON time to prevent it from occurring before the compare event (for example with a spare timer compare channel open-drain output connected with the reset signal, pulling the timer reset line down).

2.13.4 HSE/32 is not available for TIM16 input capture if RTC clock is disabled or other than HSE

Description

If the RTC clock is either disabled or other than HSE, the HSE/32 clock is not available for TIM16 input capture even if selected (bitfield TI1_RMP[2:0] = 101 in the TIM16_OR1 register).

Workaround

Apply the following procedure:

1. Enable the power controller clock (bit PWREN = 1 in the RCC_APB1ENR1 register).
2. Disable the backup domain write protection (bit DBP = 0 in the PWR_CR1 register).
3. Enable RTC clock and select HSE as clock source for RTC (bits RTCSEL[1:0] = 11 and bit RTCEN = 1 in the RCC_BDCR register).
4. Select the HSE/32 as input capture source for TIM16 (bitfield TI1_RMP[2:0] = 101 in the TIM16_OR1 register).

Alternatively, use TIM17 that implements the same features as TIM16, and is not affected by the limitation described.

2.14 LPTIM

2.14.1 Device may remain stuck in LPTIM interrupt when entering Stop mode

Description

This limitation occurs when disabling the low-power timer (LPTIM).

When the user application clears the ENABLE bit in the LPTIM_CR register within a small time window around one LPTIM interrupt occurrence, then the LPTIM interrupt signal used to wake up the device from Stop mode may be frozen in active state. Consequently, when trying to enter Stop mode, this limitation prevents the device from entering low-power mode and the firmware remains stuck in the LPTIM interrupt routine.

This limitation applies to all Stop modes and to all instances of the LPTIM. Note that the occurrence of this issue is very low.

Workaround

In order to disable a low power timer (LPTIMx) peripheral, do not clear its ENABLE bit in its respective LPTIM_CR register. Instead, reset the whole LPTIMx peripheral via the RCC controller by setting and resetting its respective LPTIMxRST bit in RCC_APByRSTRz register.

2.14.2 ARRM and CMPM flags are not set when APB clock is slower than kernel clock

Description

When LPTIM is configured in one shot mode and APB clock is lower than kernel clock, there is a chance that ARRM and CMPM flags are not set at the end of the counting cycle defined by the repetition value REP[7:0]. This issue can only occur when the repetition counter is configured with an odd repetition value.

Workaround

To avoid this issue the following formula must be respected:

{ARR, CMP} ≥ KER_CLK / (2* APB_CLK),

where APB_CLK is the LPTIM APB clock frequency, and KER_CLK is the LPTIM kernel clock frequency. ARR and CMP are expressed in decimal value.

Example: The following example illustrates a configuration where the issue can occur:

- APB clock source (MSI) = 1 MHz , Kernel clock source (HSI) = 16 MHz
- Repetition counter is set with REP[7:0] = 0x3 (odd value)

The above example is subject to issue, unless the user respects:

{CMP, ARR} ≥ 16 MHz / (2 * 1 MHz)

→ ARR must be ≥ 8 and CMP must be ≥ 8

Note: REP set to 0x3 means that effective repetition is REP+1 (= 4) but the user must consider the parity of the value loaded in LPTIM_RCR register (=3, odd) to assess the risk of issue.

2.14.3 Device may remain stuck in LPTIM interrupt when clearing event flag

Description

This limitation occurs when the LPTIM is configured in interrupt mode (at least one interrupt is enabled) and the software clears any flag in LPTIM_ISR register by writing its corresponding bit in LPTIM_ICR register. If the interrupt status flag corresponding to a disabled interrupt is cleared simultaneously with a new event detection, the set and clear commands might reach the APB domain at the same time, leading to an asynchronous interrupt signal permanently stuck high.

This issue can occur either during an interrupt subroutine execution (where the flag clearing is usually done), or outside an interrupt subroutine.

Consequently, the firmware remains stuck in the LPTIM interrupt routine, and the device cannot enter Stop mode.

Workaround

To avoid this issue, it is strongly advised to follow the recommendations listed below:

- Clear the flag only when its corresponding interrupt is enabled in the interrupt enable register.
- If for specific reasons, it is required to clear some flags that have corresponding interrupt lines disabled in the interrupt enable register, it is recommended to clear them during the current subroutine prior to those which have corresponding interrupt line enabled in the interrupt enable register.
- Flags must not be cleared outside the interrupt subroutine.

Note: The standard clear sequence implemented in the HAL_LPTIM_IRQHandler in the STM32Cube is considered as the proper clear sequence.

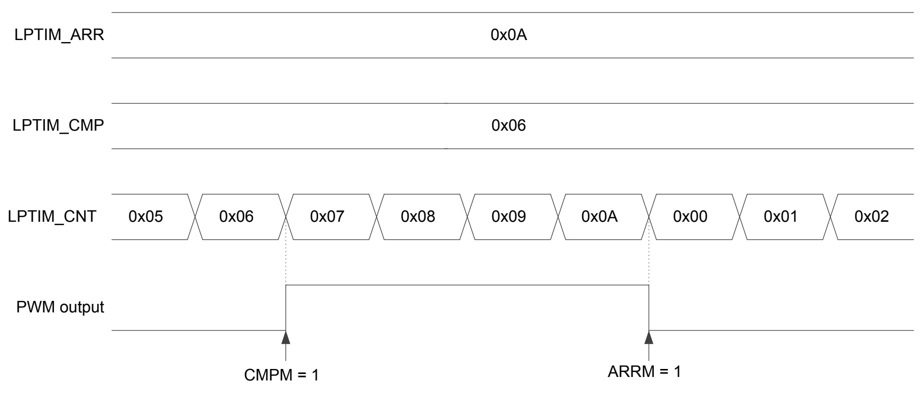
2.14.4 LPTIM events and PWM output are delayed by one kernel clock cycle

Description

The compare match event (CMPM), auto reload match event (ARRM), update event (UE), PWM output level and interrupts are updated with a delay of one kernel clock cycle.

Consequently, it is not possible to generate PWM with a duty cycle of 0% or 100%.

The following waveform gives the example of PWM output mode and the effect of the delay:



Workaround

Set the compare value to the desired value minus 1. For instance in order to generate a compare match when LPTM_CNT = 0x08, set the compare value to 0x07.

2.14.5 LPTIM1 outputs cannot be configured as open-drain

Description

LPTIM1 outputs are set in push-pull mode regardless of the configuration of corresponding GPIO outputs.

Workaround

None.

2.15 RTC and TAMP

2.15.1 Alarm flag may be repeatedly set when the core is stopped in debug

Description

When the core is stopped in debug mode, the clock is supplied to subsecond RTC alarm downcounter even when the device is configured to stop the RTC in debug.
As a consequence, when the subsecond counter is used for alarm condition (the MASKSS[3:0] bitfield of the RTC_ALRMASSR and/or RTC_ALRMBSSR register set to a non-zero value) and the alarm condition is met just before entering a breakpoint or printf, the ALRAF and/or ALRBF flag of the RTC_SR register is repeatedly set by hardware during the breakpoint or printf, which makes any attempt to clear the flag(s) ineffective.

Workaround

None.

2.15.2 A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF

Description

With the timestamp on tamper event enabled (TAMPTS bit of the RTC_CR register set), a tamper event is ignored if it occurs:

- within four APB clock cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, while the TSF flag is not yet effectively cleared (it fails to set the TSOVF flag)
- within two ck_apre cycles after setting the CTSF bit of the RTC_SCR register to clear the TSF flag, when the TSF flag is effectively cleared (it fails to set the TSF flag and timestamp the calendar registers)

Workaround

None.

2.15.3 REFCKON write protection associated to INIT KEY instead of CAL KEY

Description

The write protection of the REFCKON bit is unlocked if the key sequence is written in RTC_WPR with the privilege and security rights set by the INITPRIV and INITSEC bits, instead of being set by the CALPRIV and CALSEC bits.

Workaround

Unlock the INIT KEY before writing REFCKON.

2.15.4 Tamper flag not set on LSE failure detection

Description

With the timestamp on tamper event enabled (the TAMPTS bit of the RTC_CR register set), the LSE failure detection (LSE clock stopped) event connected to the internal tamper 3 fails to raise the ITAMP3F and ITAMP3MF flags, although it duly erases or blocks (depending on the internal tamper 3 configuration) the backup registers and other device secrets, and the RTC and TAMP peripherals resume normally upon the LSE restart.

Note: As expected in this particular case, the TSF and TSMF flags remain low as long as LSE is stopped as they require running RTCCLK clock to operate.

Workaround

None.

2.15.5 Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1

Description

When SSCLR bit of the RTC_ALRMxSSR register is set when in binary mode, SSR is reloaded with 0xFFFF FFFF at the end of the ck_apre cycle when RTC_SSR is set to RTC_ALRxBINR (*x* stands for either A or B)

RTC_SSR is not reloaded with 0xFFFF FFFF if RTC_ALRxBINR is modified while RTC_SSR is set to RTC_ALRxBINR. Rather, SSR continues to decrement.

Workaround

The workarounds are described for alarm A, and can be applied in the same manner for alarm B. Two workarounds are proposed, the second one requires to use the second alarm.

- Wait for one ck_apre cycle after an alarm A event before changing the RTC_ALRABINR register value.
- Do not reprogram RTC_ALRABINR following the alarm A event itself. Instead, use alarm B configured with RTC_ALRBBINR set to 0xFFFF FFFF, and reprogram RTC_ALRABINR after the alarm B event. This ensures that one ck_apre cycle elapses following the alarm A event.

2.15.6 RTC_REFIN and RTC_OUT on PB2 not operating in Stop 2 mode

Description

In Stop 2 low-power mode, the RTC_REFIN function does not operate and the RTC_OUT function does not operate if mapped on the PB2 pin.

Workaround

Apply one of the following measures:

- Use Stop 1 mode instead of Stop 2. This ensures the operation of both functions.
- Map RTC_OUT to the PC13 pin. This ensures the operation of the RTC_OUT function in either low-power mode. However, it has no effect to the RTC_REFIN function.

2.16 I2C

2.16.1 Wrong data sampling when data setup time ($t_{SU;DAT}$) is shorter than one I2C kernel clock period

Description

The I^{2}C-bus specification and user manual specify a minimum data setup time ($t_{SU;DAT}$) as:

- 250 ns in Standard mode
- 100 ns in Fast mode
- 50 ns in Fast mode Plus

The device does not correctly sample the I^{2}C-bus SDA line when $t_{SU;DAT}$ is smaller than one I2C kernel clock (I^{2}C-bus peripheral clock) period: the previous SDA value is sampled instead of the current one. This can result in a wrong receipt of slave address, data byte, or acknowledge bit.

Workaround

Increase the I2C kernel clock frequency to get I2C kernel clock period within the transmitter minimum data setup time. Alternatively, increase transmitter's minimum data setup time. If the transmitter setup time minimum value corresponds to the minimum value provided in the I^{2}C-bus standard, the minimum I2CCLK frequencies are as follows:

- In Standard mode, if the transmitter minimum setup time is 250 ns, the I2CCLK frequency must be at least 4 MHz.
- In Fast mode, if the transmitter minimum setup time is 100 ns, the I2CCLK frequency must be at least 10 MHz.
- In Fast-mode Plus, if the transmitter minimum setup time is 50 ns, the I2CCLK frequency must be at least 20 MHz.

2.16.2 Spurious bus error detection in master mode

Description

In master mode, a bus error can be detected spuriously, with the consequence of setting the BERR flag of the I2C_SR register and generating bus error interrupt if such interrupt is enabled. Detection of bus error has no effect on the I^{2}C-bus transfer in master mode and any such transfer continues normally.

Workaround

If a bus error interrupt is generated in master mode, the BERR flag must be cleared by software. No other action is required and the ongoing transfer can be handled normally.

2.16.3 OVR flag not set in underrun condition

Description

In slave transmission with clock stretching disabled (NOSTRETCH = 1 in the I2C_CR1 register), an underrun condition occurs if the current byte transmission is completed on the I^{2}C bus, and the next data is not yet written in the TXDATA[7:0] bitfield. In this condition, the device is expected to set the OVR flag of the I2C_ISR register and send 0xFF on the bus.

However, if the I2C_TXDR is written within the interval between two I2C kernel clock cycles before and three APB clock cycles after the start of the next data transmission, the OVR flag is not set, although the transmitted value is 0xFF.

Workaround

None.

2.16.4 Transmission stalled after first byte transfer

Description

When the first byte to transmit is not prepared in the TXDATA register, two bytes are required successively, through TXIS status flag setting or through a DMA request. If the first of the two bytes is written in the I2C_TXDR register in less than two I2C kernel clock cycles after the TXIS/DMA request, and the ratio between APB clock and I2C kernel clock frequencies is between 1.5 and 3, the second byte written in the I2C_TXDR is not internally detected. This causes a state in which the I2C peripheral is stalled in master mode or in slave mode, with clock stretching enabled (NOSTRETCH = 0). This state can only be released by disabling the peripheral (PE = 0) or by resetting it.

Workaround

Apply one of the following measures:
- Write the first data in I2C_TXDR before the transmission starts.
- Set the APB clock frequency so that its ratio with respect to the I2C kernel clock frequency is lower than 1.5 or higher than 3.

2.17 USART

2.17.1 Anticipated end-of-transmission signaling in SPI slave mode

Description

In SPI slave mode, at low USART baud rate with respect to the USART kernel and APB clock frequencies, the *transmission complete* flag TC of the USARTx_ISR register may unduly be set before the last bit is shifted on the transmit line.
This leads to data corruption if, based on this anticipated end-of-transmission signaling, the application disables the peripheral before the last bit is transmitted.

Workaround

Upon the TC flag rise, wait until the clock line remains idle for more than the half of the communication clock cycle. Then only consider the transmission as ended.

2.17.2 Data corruption due to noisy receive line

Description

In UART mode with oversampling by 8 or 16 and with 1 or 2 stop bits, the received data may be corrupted if a glitch to zero shorter than the half-bit occurs on the receive line within the second half of the stop bit.

Workaround

None.

2.18 LPUART

2.18.1 LPUART1 outputs cannot be configured as open-drain

Description

LPUART1 outputs are set in push-pull mode regardless of the configuration of corresponding GPIO outputs.

Workaround

None.

2.19 SPI

2.19.1 BSY bit may stay high when SPI is disabled

Description

The BSY flag may remain high upon disabling the SPI while operating in:

- master transmit mode and the TXE flag is low (data register full).
- master receive-only mode (simplex receive or half-duplex bidirectional receive phase) and an SCK strobing edge has not occurred since the transition of the RXNE flag from low to high.
- slave mode and NSS signal is removed during the communication.

Workaround

When the SPI operates in:

- master transmit mode, disable the SPI when TXE = 1 and BSY = 0.
- master receive-only mode, ignore the BSY flag.
- slave mode, do not remove the NSS signal during the communication.

2.19.2 BSY bit may stay high at the end of data transfer in slave mode

Description

BSY flag may sporadically remain high at the end of a data transfer in slave mode. This occurs upon coincidence of internal CPU clock and external SCK clock provided by master.

In such an event, if the software only relies on BSY flag to detect the end of SPI slave data transaction (for example to enter low-power mode or to change data line direction in half-duplex bidirectional mode), the detection fails.

As a conclusion, the BSY flag is unreliable for detecting the end of data transactions.

Workaround

Depending on SPI operating mode, use the following means for detecting the end of transaction:

- When NSS hardware management is applied and NSS signal is provided by master, use NSS flag.
- In SPI receiving mode, use the corresponding RXNE event flag.
- In SPI transmit-only mode, use the BSY flag in conjunction with a timeout expiry event. Set the timeout such as to exceed the expected duration of the last data frame and start it upon TXE event that occurs with the second bit of the last data frame. The end of the transaction corresponds to either the BSY flag becoming low or the timeout expiry, whichever happens first.

Prefer one of the first two measures to the third as they are simpler and less constraining.

Alternatively, apply the following sequence to ensure reliable operation of the BSY flag in SPI transmit mode:

1. Write last data to data register.
2. Poll the TXE flag until it becomes high, which occurs with the second bit of the data frame transfer.
3. Disable SPI by clearing the SPE bit mandatorily before the end of the frame transfer.
4. Poll the BSY bit until it becomes low, which signals the end of transfer.

Note: The alternative method can only be used with relatively fast CPU speeds versus relatively slow SPI clocks or/and long last data frames. The faster is the software execution, the shorter can be the duration of the last data frame.

2.20 SAI

2.20.1 Last SAI_MCLK clock pulse truncated upon disabling SAI master

Description

When disabling, during the communication, the SAI peripheral configured as master with the OUTDRIV bit of the corresponding SAI_xCR1 register cleared, the device may truncate the last SAI_MCLK_x bit clock pulse of the transaction, potentially causing a failure to the external codec logic.

Workaround

Set the OUTDRIV bit of the corresponding SAI_xCR1 register.

2.21 bxCAN

2.21.1 bxCAN time-triggered communication mode not supported

Description

The time-triggered communication mode described in the reference manual is not supported. As a result, timestamp values are not available. The TTCM bit of the CAN_MCR register must be kept cleared (time-triggered communication mode disabled).

Workaround

None.

2.22 OTG_FS

2.22.1 Host packet transmission may hang when connecting through a hub to a low-speed device

Description

When the USB on-the-go full-speed peripheral connects to a low-speed device via a hub, the transmitter internal state machine may hang. This leads, after a timeout expiry, to a port disconnect interrupt.

Workaround

None. However, increasing the capacitance on the data lines may reduce the occurrence.

Important security notice

Revision history

Table 6. **Document revision history**

Date	Version	Changes
16-Dec-2019	1	Initial release.
09-Mar-2020	2	Added errata in Table 3. Summary of device limitations and Section 2 Description of device errata: • Unaligned write access to OCTOSPIM configuration registers failing • Wrong ADC differential conversion result for channel 5 • Anticipated end-of-transmission signaling in SPI slave mode • Data corruption due to noisy receive line • Superseded suspend sequence for data loaded by DMA • Superseded suspend sequence for data loaded by the CPU Added Table 4. Summary of device documentation errata.
14-Jun-2021	3	Added errata in Table 3. Summary of device limitations, Table 4. Summary of device documentation errata, and Section 2 Description of device errata: • Flash memory ECC single-error correction prevents double-error detection from triggering NMI • Data corruption upon a specific FIFO write sequence to synchronous PSRAM • Memory-mapped write error response when DQS output is disabled • ADC_AWDy_OUT reset by non-guarded channels • Selected external ADC inputs unduly clamped to V_{DD} when all analog peripherals are disabled • TSC signal-to-noise concern under specific conditions • Consecutive compare event missed in specific conditions • Output compare clear not working with external counter reset • LPTIM events and PWM output are delayed by one kernel clock cycle • A tamper event fails to trigger timestamp or timestamp overflow events during a few cycles after clearing TSF • REFCKON write protection associated to INIT KEY instead of CAL KEY • Tamper flag not set on LSE failure detection • Binary mode: SSR is not reloaded with 0xFFFF FFFF when SSCLR = 1 Modified errata: • Full JTAG configuration without NJTRST pin cannot be used • DMA disable failure and error flag omission upon simultaneous transfer error and global flag clear • Device may remain stuck in LPTIM interrupt when entering Stop mode • Device may remain stuck in LPTIM interrupt when clearing event flag
09-May-2023	4	Added errata in Table 3. Summary of device limitations, Table 4. Summary of device documentation errata, and Section 2 Description of device errata: • VDIV or VSQRT instructions might not complete correctly when very short ISRs are used • Store immediate overlapping exception return operation might vector to incorrect interrupt • SRAM write error • Wrong instruction fetches from flash memory upon wakeup from Sleep or Stop mode when debug in low-power mode is enabled • Corrupted content of the backup domain due to a missed power-on reset after this domain supply voltage drop • Ports assigned to DAC cannot be used as GPIOs when the DAC feeds OPAMP • PC13 signal transitions disturb LSE • Received data corrupted after arbitration ownership toggles when using clock mode 3 and no DQS for read direction • Deadlock can occur under certain conditions • Read data corruption after partial read when crossing CSBOUND boundary • Deadlock or write-data corruption after spurious write to a misaligned address in OCTOSPI_ AR register

Date	Version	Changes
		• Read data corruption after a few bytes are skipped when crossing a four-byte boundary Modified errata: • Odd address alignment and odd byte number not supported at specific conditions • Data not sampled correctly on reads without DQS and with less than two cycles before the data phase • Memory-mapped write error response when DQS output is disabled • Byte possibly dropped during an SDR read in clock mode 3 when a transfer gets automatically split • Single-, dual- and quad-SPI modes not functional with DQS input enabled • Additional bytes read in Indirect mode with DQS input enabled when data length is too short • Device may remain stuck in LPTIM interrupt when clearing event flag

Contents